INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenen-Praktikum

Implementierung eines datenbankgestützten Workflow-Systems mit Web-Interface

David Schmitz, Bernhard Wager

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Igor Radisic
Gerald Vogt

Abgabetermin: 30. Oktober 2001

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenen-Praktikum

Implementierung datenbankgestützten Workflow-Systems mit Web-Interface

David Schmitz, Bernhard Wager

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Igor Radisic Gerald Vogt
Abgabetermin:	30. Oktober 2001

Hiermit versichern wir, dass wir das vorliegende Fortgeschrittenenpraktikum selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben.

München, den 30. Oktober 2001

. . . .
(Unterschrift der Kandidaten)

Zusammenfassung

In diesem FoPra wurde ein Workflow-System implementiert, mit dessen Hilfe Vorlesungen, Praktika und andere Lehrveranstaltungen, sowie FoPras und Diplomarbeiten vorbereitet, unterstützt und verwaltet werden können. Das System sammelt Organisationswissen und Verwaltungsdaten in einer Datenbank. Als Benutzerinterface dient das WWW. Aus der Workflow-Anwendung heraus können externe Programme gestartet werden. Die Erstellung neuer Workflows wird durch die Wiederverwendbarkeit von bereits erstellten Modulen erleichtert.

Inhaltsverzeichnis

Inhaltsverzeichnis **i**

Abbildungsverzeichnis **ii**

1 Aufgabenstellung **1**
1.1 Anforderungen an das FoPra . . . 1
1.2 Workflow-Anwendungen . . . 2

2 Architektur **4**

3 Aufbau und Erzeugung eines Workflows **6**
3.1 Einführung . . . 6
3.2 Datenbankschema der Workflow-Daten . . . 9
3.3 Wichtige Funktionen des Workflow-Systems . . . 13
3.3.1 Modulaufruf-Konzept . . . 13
3.3.2 Funktionen zum Abfragen von Status über Schritte, Subworkflow und Abhängigkeiten . . . 14
3.3.3 Funktionen zum Ändern des Zustands von Schritten bzw. Subworkflows . . . 16
3.3.4 Funktionen für Workflow-Schleifen . . . 17
3.3.5 Funktionen zum Verwalten von zusätzlichen, temporären Informationen . . . 17
3.3.6 Funktionen für Global-Actions . . . 18
3.3.7 Funktionen für die Navigation . . . 20
3.3.8 Funktionen für Workflow-Tabellen . . . 20
3.4 Funktionsweise von 'workflow.php' . . . 22
3.4.1 Standard-Aufruf und Anzeige eines Workflow-Schrittes . . . 23
3.4.2 Einzel-Aufruf eines Moduls eines Workflow-Schrittes mit eigener Darstellung 25
3.4.3 Aufruf und Darstellung einer Extrapage . . . 25
3.5 Aufbau der verschiedenen Modultypen . . . 26
3.5.1 Module vom Typ 'normal' . . . 27
3.5.2 Module vom Typ 'parent' . . . 28
3.5.3 Module vom Typ 'end' . . . 29
3.5.4 Module vom Typ 'loop' . . . 30
3.6 Anlegen eines neuen Workflows . . . 30

4 Modulprogrammierung **38**
4.1 Das Modulkonzept . . . 38
4.2 Datenbankschema . . . 39
4.2.1 Identifizierte Entities . . . 39
4.2.2 Oberklassen und relationale Datenbanken . . . 39
4.2.3 Umsetzung der ER-Diagramme in der relationalen Datenbank . . . 40
4.3 DB-Table-Submodule . . . 41
4.3.1 Einleitung . . . 41
4.3.2 Zusammensetzung des Parameter-Array $modul_param . . . 42
4.3.3 Die Einträge der Arrays $anzeige, $neues_tupel, $aendern und $loeschen im Überblick . . . 43
4.3.4 Der Eintrag query im Detail . . . 45
4.3.5 Der Eintrag cols im Detail . . . 48
4.3.6 Der Eintrag aktionen im Detail . . . 49
4.3.7 Der Eintrag table_param . . . 51
4.3.8 Der Eintrag form_param im Detail . . . 51
4.3.9 Der Eintrag appearance im Detail . . . 53
4.3.10 Der Eintrag usertext im Detail . . . 54

4.3.11 Der Eintrag prequeries im Detail 57
4.3.12 Der Eintrag composed_attribs im Detail 58
4.3.13 Der Eintrag values im Detail 59
4.3.14 Der Eintrag aendern_cols im Detail 60
4.3.15 Der Eintrag wrapper im Detail 60
4.3.16 Der Eintrag wrap_param im Detail 61
4.3.17 Der Eintrag dataconversion im Detail 63
4.3.18 Der Eintrag tabelle im Detail 64
4.3.19 Der Eintrag ziellink im Detail 65
4.4 Submodule für die Steuerung externer Module 65
4.4.1 Einleitung 65
4.4.2 Zusammensetzung des Parameter-Array $modul_param 66
4.4.3 Übergeben von Werten an ein externes Modul über die Relation workflow_extmod_param 68
4.4.4 Die Einträge 'insert' und 'check' im Detail 69
4.4.5 Der Eintrag 'new_tupel' im Detail 72
4.5 Submodule für den Aufruf externer Programme 74
4.5.1 Einleitung 74
4.5.2 Zusammensetzung des Parameter-Array $modul_param 74
4.5.3 Der Eintrag 'datei_list' im Detail 75
4.5.4 Der Eintrag 'var_format' im Detail 76
4.5.5 Der Eintrag 'system_call' im Detail 76
4.5.6 Der Eintrag 'prepare_function' im Detail 76
4.6 HTML-Schnittstelle 78
4.6.1 Funktionen zum Erstellen von Tabellen: 79
4.6.2 Funktionen die Erzeugung von Formularen 84
4.6.3 Funktionen für Erzeugung von Debugmeldungen 96
4.7 TGI als Beispiel für einen Workflow 97

5 Weitere Konzepte und Implementierungsdetails 102
5.1 Sessions 102
5.1.1 HTTP und Zustandsverwaltung 102
5.1.2 Das Session-Konzept von PHP 102
5.1.3 Das Session-Konzept der Workflow-Anwendung 103
5.2 SQL-Schnittstelle 104
5.2.1 Funktionen für die Verwaltung von Datenbank-Verbindungen 105
5.2.2 Funktionen für das Ausführen von SQL-Anfragen 106
5.2.3 Funkionen für den Zugriff auf Metadaten eines Anfrage-Ergebnisses 106
5.2.4 Funktionen für den Zugriff auf Daten des Ergebnisses einer SQL-Anfrage . 107
5.2.5 Funktionen für Informationen über Eigenschaften von Relationen 109
5.2.6 sonstige Funkionen: View- und Rechteverwaltung, etc. 110
5.3 Views und temporäre Tabellen 110

A Anhang 114
A.1 Entity-Relationship-Diagramme 114

Abbildungsverzeichnis

1 Architektur 4
2 Zugriffsmöglichkeiten 5
3 aktiver Subworkflow 6
4 inaktiver Subworkflow 7
5 strikte normale Abhängigkeit 7

6 nicht-strikte normale Abhängigkeit 7
7 normale Abhängigkeit über Subworkflow-Ebenen hinweg 8
8 Loop-Abhängigkeit 8
9 Der TGI-Workflow 98
10 ER-Diagramm für Workflow-Steuerung 114
11 ER-Diagramm für Vorlesung 115
12 ER-Diagramm für Übungsgruppe 115
13 ER-Diagramm für Übungsblatt 116
14 ER-Diagramm für Klausur 116
15 ER-Diagramm für Aufgabe 117
16 ER-Diagramm für Seminar 117
17 ER-Diagramm für Praktikum 118
18 ER-Diagramm für Praktikumsgruppe 118
19 ER-Diagramm für Fopra (Diplomarbeit) 119
20 ER-Diagramm für Schein 119

1 Aufgabenstellung

1.1 Anforderungen an das FoPra

In diesem FoPra soll ein Workflow-System implementiert werden, mit dessen Hilfe Vorlesungen, Praktika und andere Lehrveranstaltungen, sowie FoPras und Diplomarbeiten vorbereitet, unterstützt und verwaltet werden können.

Bisher war das für die Planung und Durchführung solcher Lehrveranstaltungen notwendige Organisationswissen zum Teil in Dokumenten niedergeschrieben. Solche Textdokumente existieren beispielsweise für die Durchführung von Diplomarbeiten. Für andere Veranstaltungen existieren solche Dokumente nicht. Nur der Mitarbeiter, der diese Veranstaltung schon einmal betreut hat, verfügt über das notwendige Organisationswissen. Ändert sich die Zuständigkeit innerhalb des Lehrstuhls oder verlässt ein Mitarbeiter den Lehrstuhl muss im schlimmsten Fall das Wissen komplett neu erarbeitet werden. Mit dem FoPra soll deshalb eine Möglichkeit geschaffen werden, das Organisationswissen an einer zentralen Stelle und in einer einheitlichen Form zu speichern und den zukünftig zuständigen Mitarbeitern in einer möglichst einfachen Weise zugänglich zu machen.

Da bei der Planung und Durchführung von Veranstaltungen bestimmte Reihenfolgen und Abläufe zu berücksichtigen sind, bietet es sich an das Organisationswissen in Form von Workflows zu speichern und darzustellen. Dazu werden die bei der Planung und Durchführung der Veranstaltung notwendigen Tätigkeiten in Workflow-Schritte gegliedert. Dem zuständigen Mitarbeiter wird über die Workflow-Anwendung immer der gerade aktuelle Workflow-Schritt präsentiert. Auf eventuell überfällige Schritte soll hingewiesen werden. Schritte, die von anderen nicht abgeschlossenen Schritten abhängen, sollen nicht oder nur eingeschränkt bearbeitet werden können.

Eine große Belastung für die Mitarbeiter stellen die im Zusammenhang mit einer Lehrveranstaltung notwendigen Verwaltungsarbeiten dar. So müssen beispielsweise bei der Anmeldung zu einer Vorlesung, bei der Einteilung von Übungsgruppen oder bei der Anmeldung zu einer Klausur Studentenlisten bearbeitet werden. Gerade in Hinblick auf die steigenden Studierendenzahlen ist es sinnvoll hier geeignete Hilfsmittel einzusetzen. In diesem FoPra soll deswegen zunächst eine Datenbank geschaffen werden, in der solche Daten gespeichert werden können. Die Daten sollen dann möglichst früh und möglichst durch den Verursacher der Daten in die Datenbank gelangen. Beispielsweise sollen Studenten, wenn sie an einer Veranstaltung teilnehmen wollen, ihre Daten (Name, Matrikelnummer, Haupt-, Nebenfach usw.) direkt in einen öffentlichen Bereich der Datenbank eintragen können. Die Steuerung und Kontrolle solcher Anmeldeprozeduren soll durch die Workflow-Anwendung geschehen.

Darüber hinaus soll es möglich sein, über die Workflow-Anwendung selbst Daten einzugeben und zu manipulieren. Beispielsweise könnten die studentischen Hilfskräfte für eine Vorlesung über die Workflow-Anwendung erfasst werden. Neben der reinen Präsentation des Organisationswissens soll die Workflow-Anwendung also auch verschiedene Aktionen unterstützen wie den Zugriff auf die Datenbank oder den Aufruf eines externen Programms. Dabei soll eine Anpassung bereits vorhandener Aktionen und die Integration neuer Aktionen in einen Workflow möglichst einfach sein. Bereits vorhandene Aktionen sollen in neuen Workflows wieder eingesetzt werden können. Die Wiederverwendbarkeit und Anpassbarkeit soll die Erzeugung neuer Workflows möglichst einfach gestalten. Die Erstellung und Anpassung eines Workflows soll durch ein Tool unterstützt werden. Dabei kann das Tool von der Benutzerschnittstelle her einfach gehalten sein. Die notwendigen Informationen für einen Workflow könnten beispielsweise in einer Textdatei übergeben werden.

Neben den Verwaltungsdaten, werden auch die Workflows selbst in der Datenbank abgelegt. Als Benutzerschnittstelle für das Workflow-System soll ein WWW-Browser dienen. Dazu ist eine dynamische HTML-Seiten-Generierung notwendig, wobei der aktuelle Workflow-Status, ergänzende Workflow-Informationen und eventuell darzustellende Verwaltungsdaten aus der Datenbank ausgelesen werden müssen. Im Browser vorgenommene Datenänderungen, sowie Änderungen des Workflowstatus müssen in die Datenbank zurückgeschrieben werden. In einem parallel stattfindenden FoPra wird evaluiert, welche Skriptsprache und welches relationale Datenbanksystem sich am besten dafür eignet. Unabhängig von diesen Ergebnissen wurde hier für die dynamische Erzeugung der HTML-Seiten PHP 4 verwendet. Ansonsten wurde die Implementierung so vorgenommen, dass eine möglichst große Unabhängigkeit vom zugrundeliegenden Datenbank-Management-System besteht. Direkt unterstützt werden Interbase 6, PostgreSQL 7.0 und MySQL 3.23. Eingerichtet wurde das Workflow-System am Lehrstuhl letztlich auf dem DBMS PostgreSQL 7.0.

Für das Workflow-System müssen verschiedene Benutzergruppen unterstützt werden, die sich hinsichtlich der Zugriffsrechte auf die Datenbank unterscheiden. Ein Student, der sich zu einer Vorlesung anmeldet, wird beispielsweise nur sehr eingeschränkte Zugriffsmöglichkeiten und nur auf einen öffentlichen Bereich der Datenbank haben. Die in diesem Zusammenhang auftretenden Fragen hinsichtlich der Authentifizierung sollen in dem schon erwähnten parallelen Evaluierungs-FoPra untersucht werden.

Die Anforderungen an das DB-gestützte Workflow-System noch einmal im Überblick:

- Unterstützung bei der Planung und Durchführung von Lehrveranstaltungen;
- Sammeln von Organisationswissen an einer zentralen Stelle;
- Zusammenfassen von Verwaltungsdaten in einer Datenbank;
- WWW als Benutzerinterface;
- Keine oder nur geringe Kosten;
- Schnittstelle zum Aufruf externer Programme;
- Unterschiedliche Zugriffsrechte;
- Wiederverwendbarkeit;
- Einfache Erstellung neuer Workflows.

1.2 Workflow-Anwendungen

Am Beginn des FoPras stand die Suche nach Anwendungen, die in die Entwicklung des Workflow-Systems mit einbezogen werden können. Dabei sollten natürlich möglichst keine Kosten verursacht werden und die Anwendung muss den oben genannten Kriterien genügen. Insbesondere wäre ein graphisches Tool schön, mit dessen Hilfe man Workflow-Definitionen erzeugen und anpassen kann.

Eine gute Link-Sammlung zu verschiedenen Workflow-Produkten findet man unter http://www.do.isst.fhg.de/workflow/produkte/index.html. Da die meisten der dort aufgeführten Produkte inzwischen ein Web-Interface mit zumindest eingeschränktem Zugang zum Workflow-System bieten, werden im Folgenden die Produkte nach der Art der Datenhaltung bzw. dem zugrunde liegenden DBMS unterschieden.

Die meisten großen Software-Konzerne bieten Workflow-Produkte. Zum Teil basieren diese auf einer proprietären Produktumgebung, wie z. B. Exchange von Microsoft oder Domino von Lotus. Zusätzlich gibt es Drittanbieter, die für diese proprietären Umgebungen entsprechende Produkte entwickelt haben. Beispiele sind Pavone mit dem Workflow-System Espresso, welches auf Domino von Lotus basiert, und GFI Communications mit einem Produkt für Microsofts Exchange. Bei diesen Produkten hat man in der Regel die Möglichkeit über ein Web-Interface Zugang zum Workflow-System zu erhalten. Den Standardzugang stellen allerdings die entsprechenden Clients dar, wie Notes bzw. Outlook. Die Haltung der Daten ist absolut proprietär und deshalb für unsere Zwecke nicht geeignet. Außerdem sind sie teuer.

Eine große Gruppe bilden die Produkte, die auf einem eigenständigen DBMS beruhen. So bietet z. B. IBM das Produkt MQSeries Workflow, welches auf der eignen Datenbank DB2 beruht. Von Metastorm gibt es E-Work, das auf Oracle, MS-SQL oder Access aufsetzen kann. Leider finden sich hier keine Produkte, die auf nicht-kommerziellen DBMS aufsetzen. Nur Wave-Front behauptet, dass ihr Produkt FLOWer prinzipiell auf allen SQL-fähigen DBMS läuft. (Leider bietet es kein Web-Interface.) Alle dies Produkte bringen in der Regel ein graphisches Modellierungs-Tool mit, mit dessen Hilfe man nach dem Baukasten-Prinzip ein Workflow zusammenstellen kann.

Dabei stecken z. B. bei EasyFlow von InformationsTechnologien GmbH Petri-Netze hinter der graphischen Modellierung. Auch diese Produkte sind mit gewissen Kosten verbunden.

Aus der Open-Source-Welt stammt OpenFlow, das leider erst am Anfang seiner Entwicklung steht. Es soll einmal folgende Features bieten:

- Dokumenten Management;
- Digitale Signaturen;
- Computersicherheit;
- Workflow-System;
- Netzwerk- und Internetfähigkeiten.

OpenFlow basiert auf dem Application Server Zope, der ebenfalls Opensource ist. Zope ist im wesentlichen in Python geschrieben und kann durch Python-Objekte auch erweitert werden. Zope arbeitet mit jedem gängigen Webserver zusammen, bringt aber auch einen eigenen Webserver mit. Als Betriebssysteme sind Unix und Windows-NT geeignet, Datenbanken können über SQL und ODBC angebunden werden. Neben OpenFlow gibt es vor allem "Slashdot"-ähnliche Projekte oder Community-Portalen, die mit Hilfe von Zope realisiert wurden. Zope als Basis für unser Workflow-System zu verwenden, schien uns zu aufwändig.

Da die kommerziellen Produkte auf Grund der Kosten und OpenFlow wegen der noch nicht sehr fortgeschrittenen Entwicklung als Unterstützung für das Workflow-System nicht in Frage kommen, musste die Anwendung komplett selbst entwickelt werden. Da auch kein graphisches Tool für das Erzeugen von Workflows gefunden wurde, haben wir uns dafür entschieden, für die Workflow-Definition XML-Files zu verwenden. Diese müssen von einem Perlskript abgearbeitet werden, das den Workflow in der Datenbank anlegt.

Für die Pflege und Wartung der in Betracht gezogenen, nicht kommerziellen DBMS gibt es folgende freie Projekte:

- phpMyAdmin (http://www.phpwizard.net/projects/phpMyAdmin/) für MySQL
- phpPgAdmin (http://www.greatbridge.org/project/phppgadmin) für PostgreSQL
- IbaseAdmin (http://pkedu.fbt.eitn.wau.nl/ibaseadmin/) für Interbase

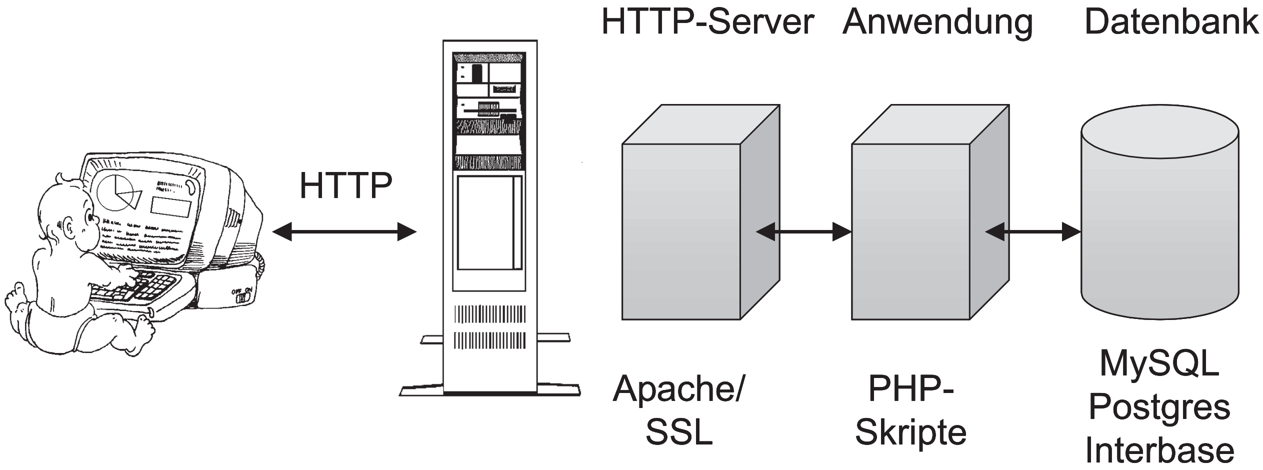

Abbildung 1: Architektur

2 Architektur

Entsprechend der Aufgabenstellung soll ein WWW-Browser als Benutzerschnittstelle zur Workflow-Anwendung dienen. Dadurch muss bei den Benutzern keine spezielle Client-Software installiert werden, da ein WWW-Browser in der Regel bei einem Betriebssystem bereits vorhanden ist oder einfach und kostenlos installiert werden kann. Um mit möglichst allen gängigen Browsern kompatibel zu sein, haben wir uns in diesem FoPra auf HTML 3.2 beschränkt. Es wurden keine Skript-Sprachen, die beim Client ausgeführt werden (z.B. JavaScript), oder JAVA-Applets verwendet. Da verschiedene Textbrowser keine Frames sinnvoll darstellen können, wurde auch auf dieses Gestaltungselement verzichtet. Letztlich war es unser Ziel, dass jeder Browser von Lynx bis Internet Explorer - auch in einer evtl. älteren Version - die generierten HTML-Seiten verarbeiten und darstellen kann.

Die Kommunikation zwischen Client und Server erfolgt auf Anwendungsebene mittels HTTP (Abbildung 1). Dabei ist zu berücksichtigen, dass dieses Protokoll zustandslos ist. Ein neuer HTTP-Request eines Clients kann allein mit den Mitteln von HTTP nicht mit vorhergegangenen Anfragen in Zusammenhang gebracht werden. Hier ist zusätzlicher Aufwand bei der Entwicklung der Anwendung notwendig, um dieses Problem zu kompensieren. Der Client spricht durch einen URL eine bestimmte Ressource auf einem Server an. Handelt es sich um ein Dokument (z.B. HTML-Dokument, Grafik), schickt der Server (entsprechende Zugriffsrechte vorausgesetzt) dieses direkt an den Client zurück. Handelt es sich um ein Skript oder ein Programm, kann er es zur Ausführung bringen. Dem Skript bzw. dem Programm können vom HTTP-Server Umgebungsvariablen übergeben werden, die u.a. vom Client zusammen mit der URL übermittelt wurden. Die Ausgabe des Programms wird dann an den Client zurückgeschickt. Im FoPra wird diese Methode verwendet, um HTML-Seiten dynamisch zu generieren.

Im einzelnen wurde als HTTP-Server Apache 1.3 mit SSL-Unterstützung unter Linux eingerichtet. Die SSL-Unterstützung ist zur Authentifizierung mittels Zertifikaten notwendig (Abbildung 2). Für die dynamische Generierung der HTML-Seiten wurde PHP (PHP Hypertext Preprocessor, früher: Personal Homepage) in der Version 4 verwendet. Hierbei handelt es sich um eine Skriptsprache, die auf dem Server ausgeführt wird. Der PHP-Code kann zusammen mit HTML-Code in einem Dokument enthalten sein, wobei der PHP-Anteil durch spezielle Tags zu kennzeichnen ist. Zur Laufzeit werden vom PHP-Interpreter die PHP-Anweisungen im Dokument verarbeitet und durch eine eventuelle Ausgabe ersetzt. Das so umgearbeitete Dokument enthält nun nur noch Code der beim Client verarbeitet werden kann, also HTML, JavaScript o.ä. Die Entscheidung für PHP wurde aus folgenden Gründen getroffen:

- C-ähnlicher Syntax (angenehm für die FoPristen);

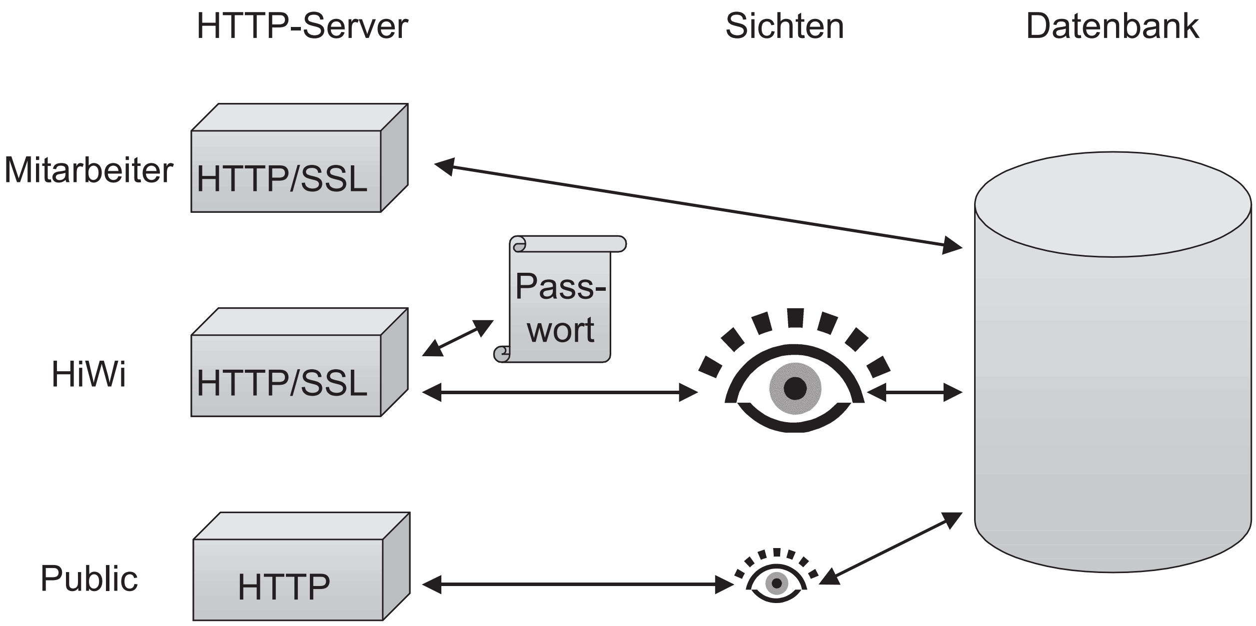

Abbildung 2: Zugriffsmöglichkeiten

- zahlreiche Bibliotheken, insbesondere für den Zugriff auf Datenbanken;
- ausreichende und aktuelle Dokumentation;
- weite Verbreitung.

Eine genaue Evaluation von PHP findet in einem anderen FoPra statt.

PHP kann klassisch über das Common Gateway Interface (CGI) oder als Webserver-Erweiterung (bei Apache: Dynamic Shared Object - DSO) benutzt werden. Die Verwendung als Webserver-Erweiterung hat den Vorteil, dass die Skripte nicht extern in jeweils eigenen Prozessen, sondern intern innerhalb des Webserver-Prozesses ablaufen. Dadurch wird eine bessere Performance erreicht.

Die PHP-Skripte greifen zur Generierung der HTML-Seiten auf Informationen zurück, die in einer Datenbank enthalten sind. Dies sind zum einen Informationen die mit dem Workflow in Zusammenhang stehen (z.B. Name des geraden aktuellen Workflowschritts), zum anderen Verwaltungsinformationen, die im Workflow dargestellt werden sollen (z.B. für eine Klausur reservierte Räume). Von PHP wird der Zugriff auf zahlreiche Datenbank Management Systeme (DBMS) durch Funktions-Bibliotheken unterstützt. Experimentiert wurde mit MySQL 3.23, PostgreSQL 7.0 und Interbase 6. Eingerichtet wurde die Datenbank letztlich unter PostgreSQL. PostgreSQL hat folgende Vorteile, durch die es sich zum Teil von den anderen DBMS unterscheidet:

- Read-only Views;
- referentielle Integrität;
- gute Unterstützung durch PHP;

MySQL hingegen kennt keine Views und auch keine referentielle Integrität. Bei Interbase ist die PHP-Unterstützung noch nicht so weit gediehen. Eine genaue Evaluation der verschiedenen DBMS findet in einem anderen FoPra statt.

3 Aufbau und Erzeugung eines Workflows

3.1 Einführung

Ein Workflow besteht aus mehreren Workflow-Schritten. Zwischen diesen bestehen verschiedene Arten von Abhängigkeiten, die festlegen, wann ein Schritt bearbeitet bzw. als abgeschlossen gekennzeichnet wird.

Die Schritte sind dabei in verschiedenen Ebenen angeordnet. Ein Schritt einer bestimmten Ebene kann dabei mehrere Schritte der darunterliegenden Ebene, die für sich gesehen einen eigenen Workflow - einen sogenannten Subworkflow - bilden, beinhalten. Ein solcher Schritt wird dann als Parent-Schritt oder kurz Parent des darunterliegenden Subworkflows bezeichnet. Der Subworkflow unter dem Parent-Schritt besteht dabei genau aus allen Schritten, die diesen Schritt als Parent-Schritt haben. Der Subworkflow kann dabei innerhalb des gesamten Workflows mehrmals durchlaufen werden, d.h. der Subworkflow ist zu einem bestimmten Zeitpunkt entweder aktiv oder inaktiv (siehe Abbildung 3 und 4). Die Steuerung der Abläufe des Subworkflows übernimmt der Parent-Schritt. In ihm wird entschieden, ob der Subworkflow unter ihm derzeit aktiviert werden muss, und es werden Parameter für den Subworkflow-Durchlauf festgelegt. Wird der Subworkflow aktiviert, so muss er i.a. selbst vollständig durchlaufen werden, bis er wieder inaktiv wird. Es besteht aber auch eine Abhängigkeit vom Subworkflow zum Parent zurück. Der Parent-Schritt selbst kann erst dann abgeschlossen werden, wenn der Subworkflow im Moment nicht mehr aktiv ist (Subworkflow-Abhängigkeit, siehe Abbildung 4). Das Subworkflow-Konzept erlaubt Rekursion, d.h. ein Schritt in einem Subworkflow kann selbst wieder Parent-Schritt eines Subworkflows sein.

Im folgenden wird der Subworkflow, in dem ein gegebener Schritt enthalten ist, der Subworkflow dieses Schrittes genannt. Davon zu unterscheiden ist der Subworkflow unter einem (Parent-)-Schritt. D.h. auch bei Parent-Schritten ist der Subworkflow dieses Parent-Schrittes der Subworkflow indem dieser enthalten ist und nicht der Subworkflow unter diesem Parent.

Zwischen Schritten eines bestimmten Subworkflows (auf der gleichen Workflow-Ebene) können nun sogenannte normale Abhängigkeiten bestehen. Ein Schritt hängt dabei von einem anderen Schritt (im folgenden Bedingungs-Schritt genannt) ab. Ein Schritt kann erst bearbeitet werden, wenn der Bedingungs-Schritt schon erledigt ist, d.h. die Abhängigkeit erfüllt ist (siehe Abbildung 5). Es besteht hierbei die Möglichkeit pro Schritt festzulegen, dass diese Bedingung gelockert wird. Dann besagt die Abhängigkeit der beiden Schritte nur, dass der abhängige Schritt erst dann erledigbar ist, d.h. als abgeschlossen markierbar ist, wenn der Bedingungs-Schritt, erledigt ist (siehe Abbildung 6). Der zuerst genannte Fall mit normalen Abhängigkeiten, die die Bearbeitungsmöglichkeit und nicht nur das als 'erledigt' Markieren eines Schrittes regeln, heißen strikte normale Abhängigkeiten.

Das Konzept der normalen Abhängigkeiten erlaubt aber nicht nur normale Abhängigkeiten zwischen Schritten des gleichen Subworkflows, sondern folgende Erweiterung: Ein gegebener Schritt kann normal abhängig von einem anderen Schritt sein, dessen Parent-Schritt (direkter

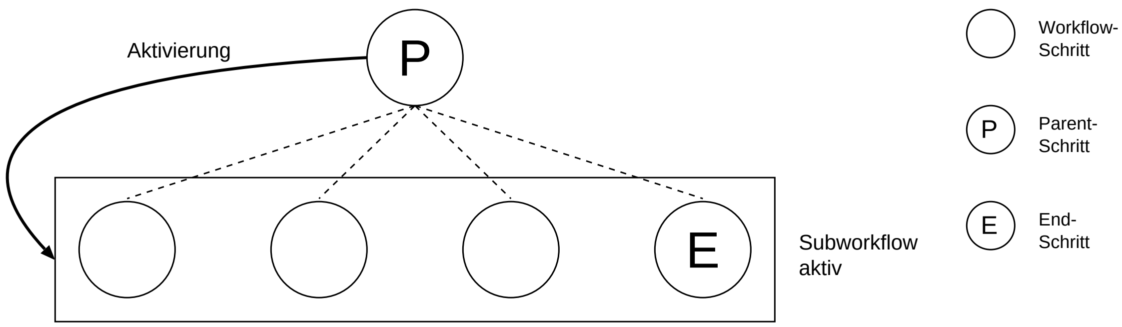

Abbildung 3: aktiver Subworkflow

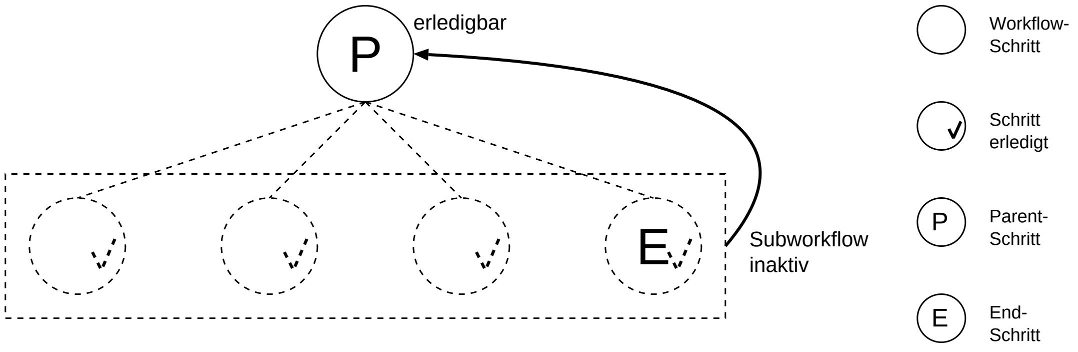

Abbildung 4: inaktiver Subworkflow

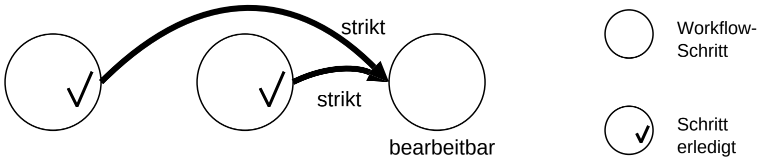

Abbildung 5: strikte normale Abhängigkeit

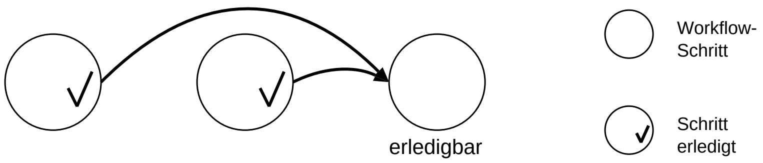

Abbildung 6: nicht-strikte normale Abhängigkeit

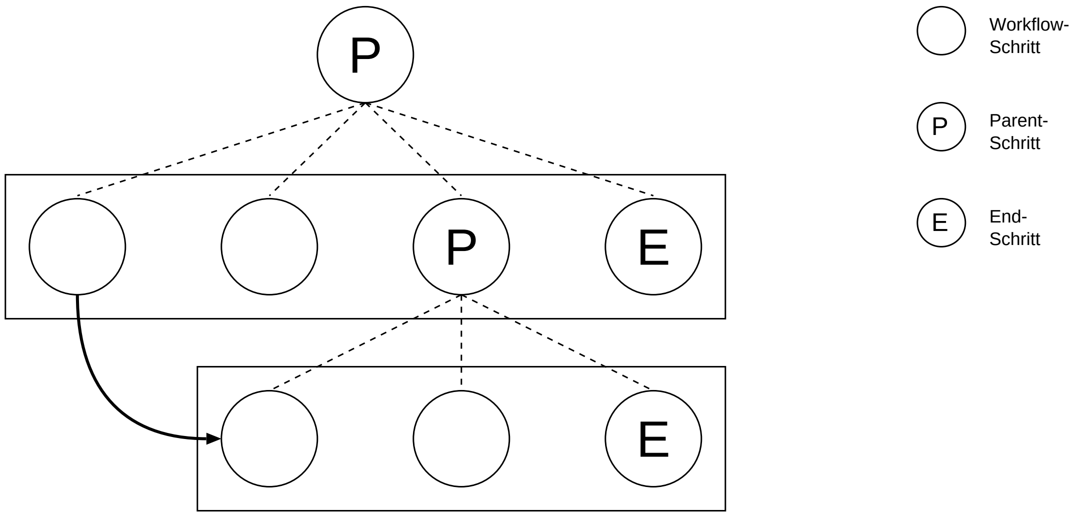

Abbildung 7: normale Abhängigkeit über Subworkflow-Ebenen hinweg

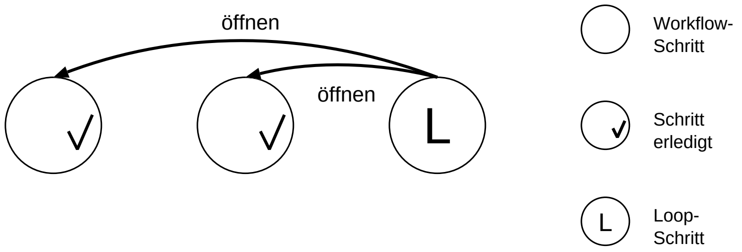

Abbildung 8: Loop-Abhängigkeit

oder) indirekter Parent-Schritt des gegebenen Schrittes ist. Dabei ist ein indirekter Parent-Schritt eines gegebenen Schrittes entweder dessen (direkter) Parent-Schritt selbst oder ein Parent-Schritt eines Schrittes, der schon ein indirekter Parent-Schritt des gegebenen Schrittes ist (parent, grandparent, grandgrandparent, ...: alle Parents rekursiv, siehe Abbildung 7). Damit schließt die neue Möglichkeit für normale Abhängigkeiten zwischen Schritten, die ursprüngliche, nämlich im gleichen Subworkflow zu sein, ein. Es ist bei dieser allgemeineren Festlegung immer noch möglich festzustellen, ob eine normale Abhängigkeit erfüllt ist oder nicht, d.h. ob der Bedingungs-Schritt erledigt oder nicht erledigt ist. Dies wäre nicht der Fall, wenn der Subworkflow, in dem der Bedingungs-Schritt enthalten ist, gerade nicht aktiv ist. Ist nämlich ein Schritt aktiv, d.h. sein Subworkflow ist aktiv, so sind alle Subworkflows, in denen die indirekten Parents liegen, aktiv, und damit auch alle Schritte in diesen Subworkflows der indirekten Parents aktiv. Also ist auch der Bedingungs-Schritt aktiv.

Als letzte Möglichkeit für eine Abhängigkeit von Schritten in einem Workflow gibt es die Loop-Abhängigkeit. Mit ihr sind Schleifen innerhalb eines Teiles eines Subworkflows (d.h. innerhalb der gleichen Workflow-Ebene) möglich. Dafür gibt es einen bestimmten Typ von Schritt, die Loop-Schritt. Von diesen aus besteht die Möglichkeit einen bestimmten Teil des Subworkflows, d.h. bestimmte, durch normale Abhängigkeiten verbundene Schritte des Subworkflows, die im aktuellen Durchlauf des Subworkflows bereits als erledigt markiert wurden, wieder als offen (als nicht vollständig erledigt, siehe Abbildung 8) und damit als neu bearbeitbar zu kennzeichnen.

Insgesamt hat damit ein Schritt im Workflow zu einem Zeitpunkt genau einen der 3 Zustände 'inaktiv', 'offen' oder 'erledigt'. Er befindet sich im Zustand 'inaktiv', wenn sein Subworkflow nicht aktiv ist. Ist sein Subworkflow aber aktiv so kann ein Schritt im Zustand 'offen' oder 'erledigt' sein. Ist er dabei im Zustand 'offen', so muss er noch bearbeitet und erledigt werden. Bearbeitet werden kann er aber erst, wenn alle seine strikten Abhängigkeiten erfüllt sind. Erst nach Bearbeitung, kann er als erledigt markiert werden und damit in den Zustand 'erledigt' versetzt werden. Dabei kann der Schritt selbst entscheiden, ab wann die Möglichkeit besteht ihn hiermit abzuschließen. Diese Art der Abhängigkeit heißt innere Abhängigkeit eines Schrittes. Das Erfüllt- oder Nicht-Erfüllt-Sein dieser inneren Abhängigkeiten wird dabei selbst vom Schritt und nicht von der globalen Workflow-Steuerung geprüft.

Zusätzlich zu normalen Schritten, d.h. Schritten die keine spezielle Steuerungsmöglichkeit für den Workflow enthalten müssen, gibt es die bereits oben genannten Parent-Schritte, die Subworkflows unter ihnen steuern, und Loop-Schritte die Schleifen realisieren, sowie in einem Subworkflow die sogenannten End-Schritte, die es ermöglichen einen Subworkflow abzuschließen, d.h. ihn wieder inaktiv zu machen, wenn er komplett durchlaufen wurde.

Jeder Schritt im Workflow-System besteht selbst aus einem oder mehreren Modulen. Jedes dieser Module ist dabei in einer eigenen php-Datei implementiert und bekommt pro Schritt und Workflow, für das es aufgerufen wird, eigene Parameter. Ähnlich wie bei den Schritten selbst gibt es meherer Typen von Modulen, nämlich 'normal', 'parent', 'loop' oder 'end'. Jeder Schritt kann dabei Module vom Typ 'normal' enthalten. Aber nur ein Parent-Schritt kann Module vom Typ 'parent', nur ein Loop-Schritt kann Module vom Typ 'loop' und nur ein End-Schritt Module vom Typ 'end' enthalten. Diese Module vom Typ 'parent', 'loop' und 'end' haben einen komplexeren Aufbau (besondere Funktionen bzw. Funktions-Rückgaben) als die Module vom Typ 'normal' (normale Module), um die Workflow-Steuerungsfunktionen ihres Schrittes zu realisieren. Daher werden verschiedene Typen von Modulen unterschieden.

Jeder Workflow ist außerdem so aufgebaut, dass er auf der obersten Ebene nur aus einem Parent-Schritt mit einem Subworkflow darunter besteht, der den gesamten Workflow steuert bzw. zur Übersicht über den ganzen Workflow dient. Dieser Parent-Schritt wird daher im folgenden top-Schritt oder Übersichts-Schritt genannt.

Außer Workflow-Schritten kann ein Workflow noch sogennante Extrapages enthalten. Diese haben keinerlei Abhängigkeiten und können zu jeder Zeit im Workflow-Ablauf aufgerufen bzw. ausgeführt werden, z.B. um allgemeine Verwaltungsoperationen zu ermöglichen. Jede Extrapage kann dabei ähnlich wie ein Workflow-Schritt aus mehreren Modulen vom Typ 'normal' aufgebaut sein.

3.2 Datenbankschema der Workflow-Daten

Im folgenden soll eine kurze Beschreibung des Datenbank-Schemas (ER-Diagramm siehe Anhang) der für die Workflows selbst benötigen Daten, d.h. Daten über Aufbau und Status des Workflows, gegeben werden. Diese Daten sind von den spezifischen Anwendungsdaten, wie z.B. Informationen über Studenten, Mitarbeiter oder auch Vorlesungen zu trennen. Hiermit sind nur Daten gemeint die für die globale Steuerung des Workflows notwendig sind, d.h. um die oben beschriebenen Konzepte zu ermöglichen. Anwendungsdaten dagegen werden nicht in der globalen Workflowsteuerung, sondern in den einzelnen Schritten selbst verarbeitet.

Ein Workflow wird in der Datenbank und auch im restlichen System eindeutig durch seine Workflownummer, eine Integer-Zahl, gekennzeichnet. Jeder Schritt innerhalb eines Workflows wird eindeutig durch seine Schrittnummer, einem String aus bis zu 30 Zeichen, gekennzeichnet. Bei Schritten wurde statt Integer-Zahlen der String als Datentyp gewählt, um bei der Bezeichnung von Schritten durch die Schrittnummer die hierarchische Strukturierung der Schritte sichtbar machen zu können. Man kann beispielsweise mit einem String aus Zahlen und dem '.' diese Strukturierung im Namen wiedergeben. Der tatsächlich verwendete Wert für Schrittnummer hat aber für das Workflow-System keine Bedeutung, sondern soll nur dem Benutzer den Umgang erleichtern. Einzige Ausnahme ist der Name des Übersichts-Schrittes, er heißt im System derzeit immer “top”.

Es folgt nun jeweils eine kurze Beschreibung der wichtigsten Datenbankrelationen:

Relation 'workflow' enthält für jeden Workflow im System einen Eintrag. Dieser besteht im wesentlichen aus der 'workflownummer', die als Schlüssel dient. Zusätzlich kann hier ein Name und eine Beschreibung des Workflows erscheinen.

Relation 'workflow_param': Hierin finden sich als Einträge die globalen Parameter der Workflows. Dies sind beim Anlegen des Workflows festgelegte Werte, die den einzelnen Schritten später zur Verfügung stehen. Ihr Schlüssel setzt sich aus der 'workflownummer' und einem bis zu 100 Zeichen langen String für den Parameternamen zusammen. Der Parameterwert ist ebenfalls als beliebiger String enthalten.

Die in dieser Relation gespeicherten Variablen sind im Workflow-System in dem globalen Array $wf_param abrufbar.

Relation 'workflow_schritt' enthält Definitionen und aktuellen Status aller Schritte aus allen Workflows. Jeder Schritt jedes Workflows ist dabei ein Eintrag. Der Schlüssel beseht aus der 'workflownummer' und der 'schrittnummer'. Weiterhin enthält ein Eintrag Attribute, die feste Eigenschaften des Schrittes beschreiben, sowie Attribute, die den aktuellen Zustand darstellen:

'workflownummer': Integer, Teil des Schlüssels

'schrittnummer': bis zu 30 Zeichen langer String, Teil des Schlüssels

'name': bis zu 250 Zeichen langer String, Name des Schrittes, der als Überschrift des Schrittes dient

'typ': String, der den Typ angibt. Er kann folgende Werte haben:

- 'Normal': Schritt ist ein normaler Schritt
- 'Subworkflow-Parent': Schritt ist Parent-Schritt mit einem Subworkflow darunter
- 'Subworkflow-End': Schritt ist End-Schritt eines Subworkflows
- 'Teilworkflow-SchleifenEnde': Schritt ist ein Loop-Schritt

'voriger_schritt': Schrittnummer des Vorgänger-Schrittes in der Webseiten-Navigation. NULL heißt es gibt keinen Vorgänger. Dies hat nichts mit Abhängigkeiten des Schrittes zu tun.

'naechster_schritt': Schrittnummer des Nachfolger-Schrittes in der Webseiten-Navigation. NULL bedeutet, es gibt keinen Nachfolger. Dies hat nichts mit Abhängigkeiten des Schrittes zu tun.

'erster_subschritt': Schrittnummer des ersten Schrittes eines Subworkflows in der Webseiten-Navigation. Wird nur bei Schritten vom Typ 'Subworkflow-Parent', also Parent-Schritten genutzt. Hat nichts mit Abhängigkeiten des Schrittes zu tun, sollte allerdings konsistenterweise auf einen Schritt zeigen, der als Parent_Schritt diesen Schritt hat.

'letzter_subschritt': Schrittnummer des letzten Schrittes eines Subworkflows in der Webseiten-Navigation. Wird nur bei Schritten vom Typ 'Subworkflow-Parent', also Parent-Schritten genutzt. Hat nichts mit Abhängigkeiten des Schrittes zu tun, sollte allerdings konsistenterweise auf einen Schritt zeigen, der als Parent_Schritt diesen Schritt hat. Sinnvoll wäre auch, das der hiermit referenzierte Schritt ein End-Schritt für den Subworkflow ist.

'parent': Schrittnummer des Parent-Schrittes dieses Schrittes, d.h. der Parent-Schritt des Subworkflows, zu dem dieser Schritt gehört. Dieses Attribut wird zur Festlegung von Parent-Subworkflow-Abhängigkeiten genutzt. Muss konsistenterweise auf einen Parent-Schritt, d.h. einen Schritt mit Typ 'Subworkflow-Parent' zeigen. Ist NULL nur für den obersten Top-Schritt eines Workflows.
Der Subworkflow unter einem Parent-Schrittes besteht genau aus den Schritten, die hier im Attribut 'parent' die Schrittnummer des Parent-Schritts stehen haben.

'erklaerung': beliebiger fester Text, der bei einem Schritt in der Webseiten-Darstellung als Erklärung angezeigt wird

'abhaengigkeiten_sind_strikt': boolscher Wert, falls 'true', so sind alle Abhängigkeiten zu diesem Schritt strikt, d.h. ist z.B. dieser Schritt von einem anderen normal abhängig, so kann erst bearbeitet werden, wenn der andere erledigt ist. Bei 'false', wäre er vor Erledigung des anderen Schrittes bereits bearbeitbar, nur nicht erledigbar.
Falls der globale Workflow-Parameter 'abhaengigkeiten_sind_strikt' in der Relation 'workflow_param' vorhanden ist, so der Wert hier ignoriert und der Wert des globalen Parameters gilt für alle Schritte.

alle folgenden Attribute sind nicht statisch, denn sie legen den aktuellen Zustand des Schrittes fest:

'status': String, der den eigentlichen Zustand des Schrittes innerhalb seines Subworkflows kennzeichnet. Mögliche Werte sind nur 'offen' oder 'erledigt'. Ist der Subworkflow dieses Schrittes nicht aktiv, so wird dieser Wert nicht verwendet.

'subworkflow_aktiv': boolscher Wert, der angibt, ob der Subworkflow unter diesem Schritt aktiv ('true') ist oder nicht ('false'). Wird nur bei Parent-Schritten verwendet, und dann nur, falls der Subworkflow diese Schrittes selbst aktiv ist.
Das bedeutet, dass ein Workflow-Schritt bzw. der Subworkflow, zu dem er gehört, genau dann aktiv ist, wenn das Flag 'Subworkflow_aktiv' bei allen Parent-Schritten (rekursiv, d.h. auch bei den indirekten Parent-Schritten, d.h. z.B. auch beim Parent-Schritt des Parent-Schrittes selbst) gleich 'true' ist.

'durchlaufnummer': Integer, der bei Parent- bzw. Loop-Schritten angibt, wie oft während des aktuellen Durchlaufs des Subworkflows unter dem Parent- bzw. die Schleife dieses Loop-Schrittes durchlaufen wurde. Diese Anzahl wird bei Neu-Aktivierung eines Subworkflows bzw. beim Zurücksetzen einer Schleife um 1 erhöht. Sie beginnt bei Neu-Aktivierung des Subworkflows unter dem Parent- bzw. des Subworkflows des Loop-Schrittes mit 0.

'erledigt_datum': SQL-Timestamp, der angibt wann der Schritt zuletzt erledigt wurde. Wird nur für Schritte die, erledigt (also aktiv) sind.

'frist_datum': SQL-Timestamp, bis zu der der Schritt erledigt sein sollte. Wird diese Frist überschritten wird in der Workflow-Anzeige eine Warnung ausgegeben.

'bemerkung': beliebiger Text, der zur Laufzeit vom Workflow aus geändert werden kann, um sich z.B. kleine Notizen zu diesem Schritt zu machen.

'tempdata_letzte_aenderung': Timestamp, der den Zeitpunkt der letzten Änderung von temporären Daten aus der Relation 'workflow_temp_data' (siehe unten) für diesen Schritt, kennzeichnet, um cachen dieser Daten zu ermöglichen.

Relation 'workflow_schritt_modul' enthält für alle Workflows jeweils alle Module eines Schrittes. Dabei besteht der Schlüssel eines Eintrags aus 'workflownummer', 'schrittnummer' und einer 'modulnummer'. 'modulnummer' ist ein Integer-Wert, um die Module eines Schrittes zu unterscheiden. Sie legt auch den Aufruf der Module innerhalb eines Schrittes fest. Module mit niedrigerer Nummer werden vor Modulen mit höherer Nummer aufgerufen. Weiterhin enthält ein Eintag als String den Modulnamen. Ist das Modul vom Typ "normal", so kann dieser Name die Form "normal:<modulname>" oder einfach "<modulname>" haben, wobei das Modul dann in der Datei "module-_<modulname>.inc.php" implementiert ist. Bei Modulen mit anderem Typ ("parent", "end", "loop") muss der Name die Form "<typ>:<modulname>" haben, wobei <typ> "parent", "end" oder "loop" ist und das Modul in der Datei "<typ>module_<modulname>.inc.php" implementiert ist.

Relation 'workflow_schritt_modul_param' enthält für jeden Schritt eines Workflows als Einträge Modul-Parameter. Der Schlüssel setzt sich aus 'workflownummer', 'schrittnummer', 'modulnummer' und 'paramnr' zusammen. Dabei ist 'paramnr' eine beliebige eindeutige Nummer. Der Variablenname ist im Attribut 'paramkey' enthalten und kann beliebig-langer Text sein. Jedoch sollte es zu einem bestimmten Wert von 'paramkey' nur einen Eintrag pro Workflow, Schritt und Modul geben. Weiterhin enthält die Relation einen beliebigen String als den Parameterwert .

Relation 'workflow_schritt_abhaengig_von' enthält alle normalen Abhängigkeiten von Schritten für alle Workflows. Jeder Eintrag beschreibt eine Abhängigkeit. Der Eintrag besteht nur aus 'workflownummer', 'schrittnummer1' und 'schrittnummer2'. Dies bedeutet, dass im jeweiligen Workflow der Schritt mit der 'schrittnummer1' normal abhängig vom Schritt mit der 'schrittnummer2' ist, d.h. Schritt 'schrittnummer2' ist Bedingungs-Schritt für Schritt 'schrittnummer1'.

Relation 'workflow_schleifenstart' beschreibt die Loop-Abhängigkeiten für alle Workflows. Ein Eintrag besteht dabei aus 'workflownummer', 'schleifenschritt' und 'schleifenstart'. Dabei ist 'schleifenschritt' die Schrittnummer des Loop-Schrittes, der bei Rücksetzung seiner Schleife beginnend beim Schritt mit der Nummer 'schleifenstart' diesen und alle seine durch normale Abhängigkeiten bestimmten Nachfolger rekursiv auf den Status 'offen' setzt. Dabei wird jedoch abgebrochen, wenn man beim Loop-Schritt selbst wieder ankommt. Pro Loop-Schritt kann es mehrere Einträge, d.h. Schleifenstarts geben.

Relation 'workflow_schritt_temp_data' ist dazu vorhanden, um dynamische Daten eines bestimmten Moduls eines Workflow-Schrittes zu speichern, die nicht schon im restlichen Workflow-Datenbankschema und auch nicht im Anwendungs-Datenbankschema erfasst sind. Diese Daten sind nur temporär und werden nur während eines Workflow-Ablaufes benötigt. Jedes Modul eines Schritt kann bzw. muss sie selbst verwalten. Beispiele hierfür sind z.B. die Nummer der aktuellen Klausur. Die Relation hat folgenden Aufbau:

Der Schlüssel setzt sich aus 'workflownummer', 'schrittnummer', 'modulnummer' und 'name' zusammen. Dabei ist 'name' ein beliebiger String aus bis zu 100 Zeichen und steht für eine Art Variablenbezeichner. Die eigentlichen Daten stehen im Attribut 'data' und können ein beliebiger String sein. Zuletzt gibt es noch das Attribut 'gueltigkeitsdauer'. Dieses sollte als Wert "global", "subworkflow" oder "workflowloop" haben. Hat es den Wert "global" so werden die Daten nie automatisch gelöscht. Bei den Werten "subworkflow" und "workflowloop" werden die Daten gelöscht, sobald der Subworkflow des Schrittes, zu dem das Modul dieser Daten gehört, neu aktiviert wird. Bei dem Wert "workflowloop" werden die Daten sogar bereits dann gelöscht, falls der erledigt Status des Schrittes, zu dem das Modul dieser Daten gehört, durch einen Loop-Schritt bei Neu-Aktivierung einer Schleife auf 'offen' gesetzt wird.

Relation 'workflow_extrapage' definiert die Extrapages aller Workflows. Sie enthält damit als Schlüssel 'workflownummer' und 'extrapagenummer'. Weiterhin kann ein String als Name vergeben werden. Der Bitvektor 'flags' beschreibt auf welchen Schrittseiten des zugehörigen Workflows Links zu der Extrapage vorhanden sind. Dabei bestimmt Bit 0, ob ein Verweis im Kopf der Übersichtsseite des Workflows (Seite des Schrittes der Parent von allen anderen ist, d.h. der Übersichts-Schritt), Bit 1, ob ein Verweis am Ende der Übersichtsseite, Bit 2, ob ein Verweis im Kopf aller Schrittseiten und Bit 3, ob ein Verweis am Ende aller Schrittseiten erscheint. Ein gesetztes Bit gibt jeweils an, dass der Link an der entsprechenden Stelle angezeigt wird.

Relation 'workflow_extrapage_modul' ist ähnlich aufgebaut wie 'workflow_schritt_modul'. Sie enthält aber für alle Workflows die Moduldefinitionen der Extrapages, und nicht der Workflow-Schritte. Die Attribute sind bis auf 'extrapagenummer', das 'schrittnummer' ersetzt, die gleichen wie in 'workflow_schritt_modul'. Bei Extrapages sind aber nur Module vom Typ "normal" zugelassen. (siehe Relation 'workflow_schritt_modul')

Relation 'workflow_extrapage_modul_param' ist ähnlich wie 'workflow_schritt_modul_param'. Hierin sind aber die Modul-Parameterdefinitionen von Extrapages und nicht von Workflow-Schritten enthalten. Bis auf das Attribut 'extrapagenummer', das 'schrittnummer' ersetzt, sind die Attribute die gleichen wie bei 'workflow_schritt_modul_param'. (siehe 'workflow_schritt_param')

Relation 'workflow_extrapage_temp_data' gibt auch Modulen in Extrapages die Möglichkeit temporäre Daten abzulegen. Sie ist ähnlich wie 'workflow_schritt_temp_data' aufgebaut. Statt 'schrittnummer' gibt es eine 'extrapagenummer'. Die anderen Attribute sind identisch mit denen von 'workflow_schritt_temp_data'. Die 'gueltigkeitsdauer' hat hier bisher keine Funktion. (siehe 'workflow_schritt_temp_data').

Im Zusammenhang mit einem laufenden Workflow stehen noch die Relationen 'workflow_view' (merkt sich welche Views angelegt sind) und 'workflow_extmod_param' (legt Parameter für Externe Module, d.h. Seiten außerhalb des Workflows, wie z.B. Studenten/Tutoren-Schnittstelle, fest). Diese werden aber in anderen Libraries verwendet und verwaltet.

3.3 Wichtige Funktionen des Workflow-Systems

In diesem Abschnitt werden einige wichtige Funktionen des Workflow-Systems und teilweise ihre Bedeutung erläutert.

3.3.1 Modulaufruf-Konzept

Der folgenden Funktions-Beschreibung sollte besondere Aufmerksamkeit geschenkt werden, um das Modulkonzept zu verstehen.

wf_call_moduls ($specialmodultyp, $step, $modulfunktion, $control_args = array(), $onlyspecial = 0, $isextrapage = 0, $modulenr = "", $noout = 0) [File 'lib-workflow-modul.inc.php']: Um jeweils die Module eines Schrittes oder einer Extrapage aufzurufen, wird die Funktion 'wf_call_moduls' aus 'lib-workflow-modul.inc.php' verwendet. Diese erhält als Parameter u.a. die Schritt- bzw. Extrapagenummer $step und ein boolsches Flag $isextrapage, um zu unterscheiden, ob die Schritt- oder die Extrapage mit Nummer $step gemeint ist. 'wf_call_moduls' bestimmt aus den Informationen der Datenbank (siehe Relationen 'workflow_schritt_modul', 'workflow_schritt_modul_param' bzw. 'workflow_extrapage_modul', 'workflow_extrapage_modul_param') welche Module auf welche Art und Weise aufgerufen werden.

Sollen nur die Module eines bestimmten Modul-Typs ('parent', 'loop' oder 'end') aufgerufen werden, so muss der Parameter $onlyspecial ungleich 'false' sein (Standart ist gleich 'false') und der Parameter $specialmodultyp den Typ-Namen ('parent', 'loop' oder 'end') enthalten. Ist $onlyspecial gleich 'false', so werden alle Module des Schrittes bzw. der Extrapage aufgerufen.

Ist andererseits der Parameter $modulnr ungleich dem leeren String, so wird nur das Modul mit der in $modulnr angegebenen Modulnummer aufgerufen.

Mit dem Parameter $modulfunktion wird angegeben, welche Funktion in den Modulen aufgerufen werden soll. Der Name der Funktion ergibt sich zu "<modultypprefix>module_<modulname>_<modulfunktion>", wobei <modultypprefix> je nach Modultyp den Wert "parent_", "loop_", "end_" oder leerer String (normales Modul) hat, <modulname> der Name des Moduls ist und <modulfunktion> den Wert von $modulfunktion enthält. D.h. der Funktionsname der aufzurufenden Funktion ergibt sich einfach zu "<moduldateiname>_<modulfunktion>", wenn <moduldateiname> der Dateiname des Moduls ohne die Endung '.inc.php' ist. Es wird hierbei erwartet, dass die Funktion in der entsprechenden Datei "<moduldateiname>.inc.php" des jeweiligen Moduls definiert wird. D.h. diese Datei wird von 'wf_call_moduls' mit dem php-Befehl 'include_once' vor dem Aufruf der Modulfunktion included. Wird im weiteren von dem Aufruf eines Moduls mit einer bestimmten Modulfunktion <modulfunktion> gesprochen, so ist mit <modulfunktion> nicht der tatsächliche php-Funktionsname aus dem Modul gemeint, sondern nur der Wert des Parameters $modulfunktion, aus dem sich der tatsächliche Name für jedes Modul, wie oben beschrieben, ableiten lässt.

Beispiele für den Wert von $modulfunktion sind "main" und "head" (siehe unten).

Die Module werden in der Reihenfolge ihrer Modulnummern aufgerufen, d.h. Module mit kleinerer Nummer zuerst.

Bei jedem Aufruf einer Funktion eines Moduls werden jeweils zwei Parameter übergeben. Als erster Parameter dient das von außen an 'wf_call_moduls' übergebene Array $control_args. Es enthält i.a. dynamische Werte, die z.B. den aktuellen Workflowschritt beschreiben, oder Variablenwerte, die von einem Parent-Schritt übergeben werden (z.B. aktuelle Klausurnummer für einen Klausur-Subworkflow). Von 'wf_call_moduls' wird in diesem noch der Parameter 'modul_nr' hinzugefügt, der die Modulnummer innerhalb des Schrittes bzw. der Extrapage des spezifischen Moduls enthält. Der zweite Parameter an die aufgerufene Funktion ist ein Array, das alle Modul-Parameter-Definitionen aus der Datenbank (siehe Relation 'workflow_schritt_modul_param' bzw. 'workflow_extrapage_modul_param') enthält. Im Gegensatz zum ersten Parameter sind diese Werte fest für ein bestimmtes Modul eines Schrittes bzw. einer Extrapage in einem Workflow nach Generierung des Workflows vorgegeben und können keine dynamischen Informationen tragen.

Falls der boolsche Parameter $noout ungleich 'false' ist, so verursacht die Funktion 'wf_call_moduls' selbst keine Ausgabe auf Stdout, um dies komplett den aufgerufenen Modulen zu überlassen. Ansonsten wird nämlich vor und nach jedem Modul-Aufruf eine HTML-Kommentar-Zeile in die Ausgabe eingebaut, die aber stören kann, falls die aufgerufenen Module die HTML-Seite vollständig selbst erzeugen (incl. html -Tag).

Die Rückgabe von 'wf_call_moduls' ist die Liste der Rückgaben der einzelnen Modul-Funktionsaufrufe.

3.3.2 Funktionen zum Abfragen von Status über Schritte, Subworkflow und Abhängigkeiten

Es existieren eine Reihe von Funktionen um die statischen bzw. dynamischen Informationen über einen Workflow-Schritt zu ermitteln. Diese Funktionen greifen dabei in der Regel auf die globalen Variablen $wf_global_steps bzw. $wf_global_steps_dyn zu, deren Inhalt beim Beginn des Aufruf des Hauptprogramms 'workflow.php' mit Werten aus der Datenbank bzw. der Session durch die Funktion 'wf_fetch_steps_info' aus 'lib-workflow-step.inc.php' initialisiert werden.

wf_step_type($step) [File 'lib-workflow-step.inc.php'] gibt für gegebene Schrittnummer $step den Typ des Schrittes, d.h. genauer das Attribut 'typ' aus der Relation 'workflow_schritt' zurück. Dieses hat einen der Werte 'Normal', 'Subworkflow-Parent', 'Subworkflow-End' oder 'Teilworkflow-SchleifenEnde'. (siehe Relation 'workflow_schritt')

wf_step_durchlaufnummer($step) [File 'lib-workflow-step.inc.php'] gibt den aktuellen Wert der Durchlaufnummer (aus Relation 'workflow_schritt') für den Schritt $step zurück. Dieser wird bei Parent- und Loop-Schritten verwendet. (siehe Relation 'workflow_schritt')

wf_step_active($step) [File 'lib-workflow-subworkflow.inc.php'] ermittelt, ob der Schritt $step aktiv ist, d.h. sein Subworkflow aktiv ist. Rückgabe ist 'true', falls der Schritt aktiv ist, sonst 'false'. Dies wird wie bei der Relation 'workflow_schritt' beschrieben rekursiv aus den Werten des Attributs 'subworkflow_aktiv' aller indirekten Parent-Schritte von $step bestimmt.

wf_step_status($step) [File 'lib-workflow-step.inc.php'] liefert einen String, der den Status eines durch $step gegebenen Workflow-Schrittes beschreibt:

"inaktiv": der Subworkflow des Schrittes ist nicht aktiv

"offen": der Subworkflow des Schrittes ist aktiv und der Schritt ist noch zu erledigen

"erledigt": der Subworkflow des Schrittes ist aktiv und der Schritt bereits erledigt

wf_subworkflow_status($step) [File 'lib-workflow-subworkflow.inc.php'] gibt einen String zurück, der beschreibt, ob der Subworkflow unter einem Parent-Schritt $step aktiv ist oder nicht:

"inaktiv": der Subworkflow unter dem Schritt $step ist nicht aktiv

"aktiv": der Subworkflow unter dem Schritt $step ist aktiv

"nonexistent": $step ist kein Parent-Schritt, d.h. kein Subworkflow darunter vorhanden

wf_parent_steps($step) [File 'lib-workflow-subworkflow.inc.php'] ermittelt die Liste aller Schrittnummern der Parent-Schritte zu einem gegebenen Schritt $step, beginnend beim direkten Parent bis zum top-Schritt.

wf_step_normal_dependencies($step) [File 'lib-workflow-dependency.inc.php'] gibt die Liste aller Schritte zurück, von denen der Schritt $step normal abhängig ist, d.h. alle Schritte, für die entsprechender Eintrag in der Relation 'workflow_schritt_abhaengig_von' existiert (siehe Relation 'workflow_schritt_abhaengig_von').

Dabei wird jedoch mittels der Funktion 'wf_check_if_valid_dependency' aus 'lib-workflow-dependency.inc.php' vorher überprüft, ob die normale Abhängigkeit wirklich sinnvoll ist, d.h. zu jedem Zeitpunkt des Workflows überprüfbar ist, d.h. ob zu jedem Zeitpunkt, wenn $step aktiv ist, auch der Schritt von dem er normal abhängt auch aktiv ist und somit einen gültigen 'erledigt'-Status("offen" oder "erledigt") hat. Nicht sinnvolle Abhängigkeiten werden nicht in die zurückgegebene Liste aufgenommen.

wf_steps_open_normal_dependencies($step) [File 'lib-workflow-dependency.inc.php'] gibt die Liste aller Schritte zurück, von denen der Schritt $step normal abhängt und die noch nicht erledigt sind, d.h. alle noch nicht erfüllten normalen Abhängigkeiten des Schrittes $step.

wf_step_has_open_dependencies($step) [File 'lib-workflow-dependency.inc.php'] berechnet ob, der Schritt $step bestimmte, nicht erfüllte Abhängigkeiten hat:

Rückgabe ist ein Liste bestehend aus 3 boolschen Werten. Der zweite bzw. dritte davon gibt an, ob externe bzw. interne Abhängigkeiten dieses Schrittes nicht erfüllt sind. Der erste ist die logische ODER-Verknüpfung des zweiten und dritten. Eine externe, nicht erfüllte Abhängigkeit meint hier, dass der Schritt $step nicht aktiv oder unerfüllte normale Abhängigkeiten besitzt. Eine innere, nicht erfüllte Abhängigkeit meint hier nur, dass $step ein Parent-Schritt mit einem Subworkflow darunter ist, der momentan aktiv ist, also $step im Moment nicht als 'erledigt' markierbar ist.

3.3.3 Funktionen zum Ändern des Zustands von Schritten bzw. Subworkflows

wf_subworkflow_open($parentstep) [File 'lib-workflow-subworkflow.inc.php'] öffnet, d.h. aktiviert, einen bisher inaktiven Subworkflow unter gegebenem Parent-Schritt $parentstep. D.h. setzt das Attribut 'subworkflow_aktiv' des Schrittes $parentstep auf 'true', setzt von allen Schritten des Subworkflows unter $parentstep den 'erledigt'-Status auf 'offen', erhöht das Attribut 'durchlaufnummer' von $parentstep um 1 und löscht alle temporären Daten aus der Relation 'workflow_temp_data' die als 'gueltigkeitsdauer' einen Wert "subworkflow" oder " workflowloop" haben. Die Funktion macht dabei alle relevanten Änderungen in der Datenbank und in der Session.

Um Parent-Module dieses Öffnen des Subworkflows anzuzeigen, wird in der globalen Variable $wf_notify der Parameter 'subworkflow_has_opened' auf '1' gesetzt.

Anfangs wird jedoch überprüft, ob $parentstep ein Parent-Schritt ist, ob $parentstep den 'erledigt'-Status 'offen' hat und ob der Subworkflow noch nicht aktiv ist. Ist dies nicht der Fall so wird abgebrochen.

wf_subworkflow_close($step) [File 'lib-workflow-subworkflow.inc.php'] schließt, d.h. deaktiviert', einen bisher aktivierten Subworkflow unter gegebenem Parent-Schritt $step oder einem End-Schritt $step des Subworkflows (abhängig vom Typ von $step). Dabei wird das Attribut 'subworkflow_aktiv' des Parent-Step über dem Subworkflow auf 'false' gesetzt. Alle nötigen Änderungen in der Datenbank bzw. in der Session werden hier durchgeführt.

Um Parent-Module diese Schließung des Subworkflows anzuzeigen, wird in der globale Variable $wf_notify der Parameter 'subworkflow_has_closed' auf '1' gesetzt.

Zu Beginn wird jedoch überprüft, ob $step Parent-Schritt über einem Subworkflow bzw. End-Schritt eines Subworkflows ist, ob $step den Status "offen" hat und ob der so spezifizierte Subworkflow aktiv ist. Falls nicht wird der Funktions-Aufruf abgebrochen.

wf_step_close($step) [File 'lib-workflow-step.inc.php'] markiert den Schritt $step als 'erledigt', d.h. setzt das Attribut 'status' für $step in der Relation 'workflow_schritt' auf 'erledigt' und setzt das 'erledigt_datum' für $step auf die aktuelle Uhrzeit. Alle notwendigen Änderungen werden in Datenbank und Session durchgeführt.

Diese Funktion wird nur der Vollständigkeit halber hier aufgeführt und sollte in Modulen nicht unbedingt aufgerufen werden, da dies eine Aufgabe der globalen Workflow-Steuerung ist.

wf_step_open($step) [File 'lib-workflow-step.inc.php'] setzt den 'erledigt'-Status, d.h. das Attribut 'status' in 'workflow_schritt' des Schrittes $step auf "offen". Datenbank und Session werden entsprechend angepasst.

Diese Funktion ist nicht unbedingt dazu gedacht, in einem Modul aufgerufen zu werden, da der 'erledigt'-Status eines Schrittes von der globalen Workflow-Steuerung gesteuert wird.

wf_get_frist_datum($step) [File 'lib-workflow-step.inc.php'] liefert den Frist-Zeitpunkt, bis zu dem der Schritt $step erledigt sein sollte, d.h. den Zeitpunkt, ab dem einen Warnung angezeigt wird, falls $step noch nicht erledigt ist. Diese Zeitinformation wird in Form des sogenannten wftime-Formats zurückgegeben. Dies ist einfach ein Array, dass i.a. die fast selbsterklärenden Parameter 'year', 'month', 'mday', 'hour', 'min' und 'sec' enthält (Speziell hier ist aber keine Uhrzeit enthalten). Dabei ist die Jahreszahl aus dem Parameter 'year' 4-stellig (z.B. 1950, 2000, ...) und die Werte für 'month' bzw. 'mday' die Nummer des Monats bzw. des Tages innerhalb eines Monats, gerechnet von 1. (siehe 'lib-time.inc.php'). Änderungen finden in Datenbank und Session entsprechend statt.

wf_set_frist_datum($step, $datum0, $format) [File 'lib-workflow-step.inc.php'] setzt den Frist-Zeitpunkt, d.h. das Attribut 'frist_datum' in der Relation 'workflow_schritt', für den

Schritt $step. Die Änderungen finden in Datenbank und Session sofort statt. Dabei enthält $datum0 die Zeitangabe und $format gibt an, in welcher Form sie beschrieben ist. Es gibt zwei mögliche Formate:

"hash": Das Datum $datum0 ist im wftime-Format (siehe 'wf_get_frist_datum' oben).

"string": Das Datum $datum0 ist ein String, z.B. in der Form "23.8.2005".

3.3.4 Funktionen für Workflow-Schleifen

wf_steps_in_loop($loopendstep) [File 'lib-workflow-loop.inc.php'] ermittelt die Liste der Schrittnummern aller Workflow-Schritte in einer Schleife des Loop-Schrittes $loopendstep. D.h. zunächst werden aus der Relation 'workflow_schleifenstart' für $loopendstep die Schrittnummern aller Anfänge der Schleife bestimmt. Von diesen aus werden anhand ihrer normalen Abhängigkeiten weitere Mitglieder-Schritte der Schleife ermittelt. Jeder Schritt, der von einem Schritt, der bereits als Mitglied der Schleife erkannt wurde, normal abhängt, wird auch als Mitglied der Schleife betrachtet. Ausnahme bildet nur der Loop-Schritt $loopendstep selbst, er wird nicht in die Liste der Schleifenmitglieder aufgenommen, denn er stellt das Ende der Schleife dar.

wf_loop_iterate($loopendstep) [File 'lib-workflow-loop.inc.php'] führt einen neuen Durchlauf der Schleife des Loop-Schrittes $loopendstep aus. Dabei wird der 'erledigt'-Status von allen Mitgliedern der Schleife wieder auf 'offen' gesetzt, diese Schritte also als nicht erledigt markiert, so dass sie von neuem bearbeitbar sind. Außerdem wird das Attribut 'durchlaufnummer ' der Relation 'workflow_schritt' für $loopendstep um 1 erhöht und von allen Schritten der Schleife werden temporäre Daten aus der Relation 'workflow_temp_data', die eine 'gueltigkeitsdauer' gleich 'workflowloop' haben, gelöscht. Die Funktion macht hierbei alle relevanten Änderungen in der Datenbank und in der Session.

Zu Beginn wird jedoch geprüft, ob $loopendstep überhaupt ein Loop-Schritt ist und den 'erledigt'-Status 'offen' hat. Ist dies nicht der Fall, so wird abgebrochen.

3.3.5 Funktionen zum Verwalten von zusätzlichen, temporären Informationen

Die folgenden Funktionen dienen dem Zugriff und der Verwaltung der temporären Informationen der Workflow-Schritte, die in der Relation 'workflow_schritt_temp_data' gespeichert sind (bzw. in der Relation 'workflow_extrapage_temp_data' für Extrapages). Workflow-Schritte (bzw. Extrapages) können hiermit zusätzliche Informationen ablegen, die mit dem Workflow-Ablauf in Zusammenhang stehen, aber nicht in den sonstigen Workflow-Datenbank-Relationen enthalten sind. Dabei sind die Schritte für die Benutzung und Verwaltung dieser Daten selbst verantwortlich (siehe Relationen 'workflow_temp_data' bzw. 'workflow_extrapage_temp_data'). Für den tatsächlichen Zugriff auf diese Datenbank-Relationen sollten aber, vor allem für das Löschen und Ändern aus Konsistenzgründen, die folgenden Funktionen für die Verwaltung verwendet werden.

wf_temp_data_get($isextrapage, $step, $modulenr="" , $name="") [File 'lib-workflow-tempdata.inc.php'] liest bestimmte temporäre Daten des Schrittes mit der Schrittnummer $step bzw. der Extrapage mit der Nummer $step (abhängig davon, ob $isextrapage 'false' oder' true ist) aus der Relation 'workflow_temp_data' bzw. 'workflow_extrapage_temp_data'. Ist $modulnummer nicht gleich dem leeren String, so werden nur die Daten des Schritt-Moduls mit dieser Nummer gelesen. Wurde der Variablenname $name angegeben (ist nicht gleich dem leeren String), so werden nur die Variablen mit Namen $name der einzelnen Module des Schrittes gelesen. Zurückgegeben wird eine Liste von Datenbank-Ergebnistupeln (wie von 'db_fetchall_array' geliefert), die als Attribute 'modulnummer', 'name', 'data' und 'gueltigkeitsdauer' der entsprechenden Datenbank-Relation enthält (siehe Relationen 'workflow_temp_data' bzw. 'workflow_extrapage_temp_data').

wf_temp_data_get_value($isextrapage, $step, $modulenr, $name) [File 'lib-workflow-tempdata.inc.php'] liest ähnlich wie 'wf_temp_data_get' temporäre Daten. Hat die gleichen Parameter. Allerdings wird hier nur genau von dem Modul mit Nummer $modulnummer die Variable mit Name $name gelesen und nur ihr Inhalt zurückgeliefert.

wf_temp_data_del($isextrapage, $step, $modulnr="", $name="", $gueltigkeit="") [File 'lib-workflow-tempdata.inc.php'] löscht bestimmte temporäre Daten des Schrittes mit der Schrittnummer $step bzw. der Extrapage mit der Nummer $step (abhängig davon, ob $isextrapage 'false' oder' true ist) aus der Relation 'workflow_temp_data' bzw. 'workflow_extrapage_temp_data'. Ist $modulnummer nicht gleich dem leeren String, so werden nur die Daten des Schritt-Moduls mit dieser Nummer gelöscht. Wurde der Variablenname $name angegeben (ist nicht gleich dem leeren String), so werden nur die Variablen mit Namen $name der einzelnen Module des Schrittes gelöscht. Ist mit $gueltigkeit eine Liste von Gültigkeitsdauern (wie z.B. "subworkflow" oder "workflowloop", vgl. Attribut 'gueltigkeitsdauer' in Relation 'workflow_temp_data') angegeben, so werden nur die Daten gelöscht, die deren 'gueltigkeitsdauer' einen Wert aus der List $gueltigkeit hat. Enthält $gueltigkeit nur ein Element, kann dieser auch statt einer Liste einzeln in $gueltigkeit übergeben werden.

Zum Löschen von Daten aus 'workflow_temp_data' bzw. 'workflow_extrapage_temp_data' sollte stets diese Funktion verwendet werden und nie direkt gelöscht werden. Denn der Inhalt der Relation 'workflow_temp_data' wird aus Performance-Gründen in der Session gecachet. Pro Workflow-Schritt wird in der Relation 'workflow_schritt' mit dem Attribut 'tempdata_letzte_aenderung' ein Timestamp gespeichert, der den Zeitpunkt der letzten Änderung der temporären Daten dieses Schrittes kennzeichnet. 'w_temp_data_del' setzt diesen automatisch.

wf_temp_data_set($isextrapage, $step, $modulnr, $name, $data, $gueltigkeit) [File 'lib-workflow-tempdata.inc.php'] setzt bzw. ändert eine gegebene temporäre Variable des Schrittes mit der Schrittnummer $step bzw. der Extrapage mit der Nummer $step (abhängig davon, ob $isextrapage 'false' oder' true ist) aus der Relation 'workflow_temp_data' bzw. 'workflow_extrapage_temp_data'. Dabei ist $modulnummer die Nummer des Moduls im Schritt, $name ist der Name der Variable, $data der Inhalt der Variable und $gueltigkeit ist die 'Gültigkeitsdauer' (siehe Attribut 'gueltigkeitsdauer' von 'workflow_temp_data') mit der die Information gespeichert wird.

Zum Ändern von Daten aus 'workflow_temp_data' bzw. 'workflow_extrapage_temp_data' sollte stets diese Funktion verwendet werden und nie direkt verändert werden. Denn der Inhalt der Relation 'workflow_temp_data' wird aus Performance-Gründen in der Session gecachet. Pro Workflow-Schritt wird in der Relation 'workflow_schritt' mit dem Attribut 'tempdata_letzte_aenderung' ein Timestamp gespeichert, der den Zeitpunkt der letzten Änderung der temporären Daten dieses Schrittes kennzeichnet. Dieser wird hier automatisch angepasst.

3.3.6 Funktionen für Global-Actions

Will ein Modul vom Typ "parent", "end" oder "loop" eine bestimmte Workflow-Steuerungs-Aktion, wie z.B. das Öffnen eines Subworkflows, mit Hilfe der oben beschriebenen Funktionen ausführen, so kann es diese Funktionen direkt aufrufen. Das hat aber den Nachteil, dass der Workflow-Zustand erst nach dem Aufruf der jeweiligen Funktion durch das Modul geändert wird. Möglicherweise haben zuvor bereits andere Module oder die globale Workflow-Steuerung eine HTML-Ausgabe erzeugt, die nicht konsistent zum neuen Zustand ist. Damit alle Module den gleichen Workflow-Zustand betrachten, bedarf es einer Möglichkeit, bereits vor dem Aufbau der HTML-Seite bestimmte Workflow-Steuerungs-Aktionen zu veranlassen. Dies wird durch das Prinzip der sogenannten Global-Actions möglich gemacht. Steuer-Module (Typ "parent", "end" oder "loop") müssen hierbei nicht eigene Forms oder Links erzeugen, die sie beim nächsten php-Aufruf selbst verarbeiten. D.h. die Module müssen die entsprechende Aktion nicht selbst auslösen, sondern die Workflow-Steuerungs-Aktion wird zu Beginn des php-Aufrufes durch die Funktion 'wf_do_globalactions' aus 'lib-workflow-globalaction.inc.php' ausgeführt, bevor eine Workflow-Zustands-relevante HTML-Ausgabe erzeugt wird. Die hierfür notwendigen Funktionen sind diese:

wf_globalaction_link_parameter($action, $step) [File 'lib-workflow-globalaction.inc.php'] gibt einen Parameter-String zurück, der als Teil eines Parameterstrings eines HTML-Link für das Workflow-System entsprechende Variablen setzt, die eine durch $action und $step bestimmte Global-Action bei Beginn des php-Aufrufes vor Erzeugung anderer HTML-Ausgabe auslösen.

Dabei wird die eigentliche Aktion von $action festgelegt und $step wird i.a. als Parameter für diese verwendet. Möglich Werte für $action sind diese:

"subworkflow_open": veranlasst Aufruf von 'wf_subworkflow_open($step) ', d.h. Neu-Aktivierung des Subworkflow unter Parent-Schritt $step

"subworkflow_close": veranlasst Aufruf von 'wf_subworkflow_close($step)', d.h. De-Aktivierung des Subworkflows unter Parent-Schritt $step bzw. des Subworkflows des End-Schrittes$step

"step_open": veranlasst Aufruf von 'wf_step_open($step)', d.h. 'erledigt'-Status des Schrittes $step auf 'offen' setzen

"step_close": veranlasst Aufruf von 'wf_step_close($step)', d.h. 'erledigt'-Status des Schrittes $step auf 'erledigt' setzen

"loop_iterate": veranlasst Aufruf von 'wf_loop_iterate($step)', d.h. Neu-Durchlauf der Schleife des Loop-Schrittes $step

wf_globalaction_form_element($action, $step) [File 'lib-workflow-globalaction.inc.php'] erstellt ähnlich wie 'wf_globalaction_link_parameter' HTML-Code, um eine durch $action und $step bestimmte Global-Action vorzubereiten. Allerdings wird hier ein hidden-Form-Element erzeugt, das in einem Formular für das Workflow-System, bei Aufruf des Formulares die für das System relevanten Variablen setzt, um die Global-Action auszulösen. $action und $step haben die gleiche Bedeutung wie bei 'wf_globalaction_link_parameter'.

Die Rückgabe ist ein String mit dem HTML-Code für die hidden-Elemente, der ggf. vom Funktions-Aufrufer selbst noch mit dem php-Befehl 'echo' ausgegeben werden sollte.

wf_temp_data_set_form_element($isextrapage, $step, $modulenr, $name, $data, $gueltigkeit, $formparam = array()) erzeugt HTML-Code für ein Form-Element, das in einem Formular für das Workflow-System, bei Aufruf des Formulars für das System nötige Variablen setzt, die bei Beginn eine Global-Action auslösen, die eine temporäre Variable in der Relation 'workflow_schritt_temp_data' bzw. 'workflow_extrapage_temp_data' ändert. Die Global-Action veranlasst den Aufruf von 'wf_temp_data_set($isextrapage, $step, $modulenr, $name, $data, $gueltigkeit)' , also dass setzen der temporären Varibale $name des Moduls $modulenr des Schrittes $step bzw der Extrapage $step (abh. davon, ob $isextrapage 'false' oder 'true') auf den Wert $data mit der Gültigkeitsdauer $gueltigkeit (siehe 'wf_temp_data_set' oben). Mit dem Parameter-Array $formparam kann die Art des zu verwendenden Form-Elmentes definiert werden (siehe 'html_form'-Funktionen von 'lib-html.inc.php'). Als Default wird ein hidden-Form-Element verwendet. Die Rückgabe ist ein String mit dem HTML-Code für das Form-Element.

wf_notify($step) [File 'lib-workflow-step.inc.php'] liefert das globale Parameter-Array $wf_notify für den Schritt $step zurück. Wird z.B. im Zusammenhang mit Global-Action-Änderungen verwendet, um Parent-Modulen mitzuteilen, dass sich der unter ihnen liegende Subworkflow im aktuellen Lauf von 'workflow.php' verändert hat, so dass diese evtl. weitere, notwendige modulspezifische Aktionen durchführen können.

3.3.7 Funktionen für die Navigation

Folgende Funktionen sind für die Navigation im Workflow-System gedacht. Sie erzeugen HTML-Quellcode für einen Link zu einem bestimmten Schritt oder einer bestimmten Extrapage:

wf_step_link ($step, $text="", $myparam=array()) [File 'lib-workflow-navigation.inc.php'] gibt HTML-Code für einen Link zum Schritt $step im Normal-Modus ($param[wf_step_position]="Main", siehe 3.4) aus. Ist $text der leere String so wird Name und Nummer von $step als Beschriftung des Links gewählt, sonst $text. Mit dem Array $myparam kann man in den Link zusätzlich gegebene Werte-Belegungen für das global Array $param aufnehmen (vgl. 3.4).

Der Link-Code wird als String zurückgegeben und muss vom Aufrufer der Funktion selbst weiterverarbeitet werden (z.b. mit dem php-Befehl 'echo' ausgegeben werden).

wf_step_fullpage_link($step, $module_nr, $text, $function="", $myparam=array()) [File 'lib-workflow-navigation.inc.php'] erstellt ähnlich wie 'wf_step_link' HTML-Code für einen Link zum Schritt $step. Allerdings wird der Schritt so aufgerufen, dass nur sein Modul mit Nummer $module_nr mit der Modulfunktion $function (Default ist hierbei "fullpage") aufgerufen wird, die selbst den gesamten Quellcode der HTML-Seite erzeugen muss (siehe $param[wf_step_position]="ModulPage" in 3.4). Der generierte HTML-Code wird wie bei 'wf_step_link' als String zurückgegeben.

wf_extrapage_link($nr, $text="", $myparam=array()) [File 'lib-workflow-navigation.inc.php'] erzeugt HTML-Code für einen Link zu einer Extrapage. Dabei gibt $nr die Nummer der Extrapage an. Die Beschriftung kann in $text vorgegeben werden. Ist diese Vorgabe der leere String, so werden Name und Nummer der Extrapage als Beschriftung gewählt. Mit dem Array $myparam kann man in den Link zusätzlich gegebene Werte-Belegungen für das global Array $param aufnehmen (vgl. 3.4).

Der Link-Code wird als String zurückgegeben und muss vom Aufrufer der Funktion selbst weiterverarbeitet (z.b. mit dem php-Befehl 'echo' ausgegeben werden).

wf_extrapage_fullpage_link($nr, $module_nr, $text, $function="", $myparam=array()) [File 'lib-workflow-navigation.inc.php'] erstellt wie 'wf_extrapage_link' HTML-Code für einen Verweis zu der Extrapage mit Nummer $nr. Allerdings wird die Extrapage so aufgerufen, dass nur sein Modul mit Nummer $module_nr mit der Modulfunktion $function (Default ist hierbei "fullpage") aufgerufen wird, die selbst den gesamten Quellcode der HTML-Seite erzeugen muss (siehe $param[wf_extrapage_position]="ModulPage" in 3.4). Der generierte HTML-Code wird wie bei 'wf_extrapage_link' als String zurückgegeben.

3.3.8 Funktionen für Workflow-Tabellen

Die hier beschriebenen Funktionen erzeugen HTML-Code für eine tabellarische Darstellung des Zustands von Workflow-Schritten.

wf_table ($parent, $recursive, $table_param = array(), $condition_tableparam_list = array(), $with_head_and_end = 1) [File 'lib-workflow-table.inc.php']: erstellt HTML-Code für eine Tabelle, in der pro Zeile Informationen über den Zustand eines Schrittes des Workflows angezeigt wird (bzw. Formulare miteingebunden werden können, um bestimmte Änderungen zu ermöglichen). Dabei kann festgelegt werden, über welche Schritte welche Information wie angezeigt wird.

Die Festlegung, welche Schritte in der Tabelle vorkommen, geschieht mit den Parametern $parent, $recursive. Und zwar werden genau die Schritte angezeigt, die den Parent-Schritt $parent als direkten und oder indirekten Parent haben, je nachdem $recursive 'false' oder 'true' ist. D.h. diese Funktion zeigt die Schritte eines Subworkflows auf einer bestimmten Ebene oder sogar alle

Schritte eines Subworkflows (rekursiv) an. Die Reihenfolge der Schritte wird dabei durch die Navigations-Reihenfolge (d.h. das Attribut 'naechster_schritt' aus 'workflow_schritt') bestimmt.

Die Zeilen der HTML-Tabelle werden hierbei durch die 'html_table'-Funktionen aus 'lib-html.inc.php' erstellt (siehe Dokumentation 'lib-html'). Dabei wird an die Funktion 'html_table_line' stets als Parameter $cols eine feste Liste aus den Werten "schrittnummer", "namensfeld", "status", "info", "datumsfeld", "erklaerung" und "bemerkung" (in dieser Reihenfolge) übergeben. Damit werden die Tabellenspalten festgelegt: Jeweils eine Spalte für die Schrittnummer, den Namen des Schrittes (Attribut 'name' aus Relation 'workflow_schritt'), den Status ('inaktiv', 'offen', 'erledigt'), weitere Statusinformation (z.B. besonderer Schritt-Typ oder Subworkflow-Status bei Parent-Schritten), Datumsinformation (entweder Erledigungs-Frist oder der Zeitpunkt, als der Schritt als 'erledigt' markiert wurde), die Erklärung (Attribut 'erklaerung' in Relation 'workflow_schritt', fest vorgegeben) und die Bemerkung (Attribut 'bemerkung' aus Relation 'workflow_schritt') des Schrittes.

Das Parameter-Array $table_param dient als Parameter $param für die Funktion 'html_table_line'. Vorher wird er jedoch von 'wf_table' bearbeitet. Ganz zu Beginn werden einige Standardwerte vorgegeben, falls sie nicht spezifiziert wurden (siehe globale Variable $wf_table_defaults am Anfang von 'lib-workflow-table.inc.php') .

Die Funktion 'html_table_line' erhält als übergebenes Datentupel, eine Array das folgenden Information über jeweiligen Schritt enthält:

schrittnummer: Schrittnummer des anzuzeigenden Schrittes, gedacht für Tabellen-Spalte "schrittnummer"

name: Name des Schrittes, d.h. Attribut 'name' aus Relation 'workflow_schritt'

namensfeld: HTML-Link, der zum Schritt verweist und als Beschriftung dessen Namen trägt , gedacht als Inhalt der Spalte "namensfeld"

status: "inaktiv", "offen" oder "erledigt" (wie von Funktion 'wf_step_status' zurückgegeben'), gedacht als Inhalt der Spalte "status"

info: abhängig vom Schritt-Typ, gedacht als Inhalt der Spalte "info"

frist_datum: Frist, bis zu der der schritt erledigt sein sollte, d.h. Attribut 'frist_datum' aus Relation 'workflow_schritt' in lesbaren String umgewandelt

erledigt_datum: Zeitpunkt, zu der der Schritt als 'erledigt' gekennzeichnet wurde, d.h. Attribut 'erledigt_datum' aus Relation 'workflow_schritt' in lesbaren String umgewandelt

datumsfeld: je nachdem, ob der Schritt 'offen' oder 'erledigt' ist, die Frist, bis zu der er erledigt sein sollte oder der Zeitpunkt, als er als 'erledigt' markiert wurde, d.h. entweder identisch mit dem Parameter 'frist_datum' oder dem Parameter 'erledigt_datum', gedacht als Inhalt der Spalte "datumsfeld".
Auch der Parameter 'data_bgcolor' in $table_param für die Farbe der Spalte "datumsfeld" wird pro Schritt mit Hilfe der zusätzlichen Parameter 'data_bgcolor_Inaktiv', 'data_bgcolor_Offen', 'data_bgcolor_FristAbgelaufen' und 'data_bgcolor_Erledigt' (je nach Status des Schrittes) angepasst.

erklaerung: Attribut 'erklaerung' aus Relation 'workflow_schritt' für die Spalte "erklaerung"

bemerkung: Attribut 'bemerkung' aus Relation 'workflow_schritt' für die Spalte "bemerkung"

Darüberhinaus enthält das Datentupel die Variablen 'typ', 'voriger_schritt', 'naechster_schritt', 'erster_subschritt', 'letzter_subschritt', 'parent_schritt', 'abhaengigkeiten_sind_strikt', 'subworkflow_aktiv', und 'durchlaufnummer', die den übrigen Attributen der Relation 'workflow_schritt' entsprechen. Diese könnten bei Bedarf in die Darstellung (über $table_param[datacell_format]) verwendet werden.

Um verschiedene Schritte verschieden anzuzeigen, dient der Parameter $condition_table_param. Dieser ist eine Liste von sogenannten bedingten Table-Parameter-Arrays, d.h. jedes Listenelement ist ein Array, das die Parameter 'condition_type', 'condition_arg' und 'table_param' enthält. Dabei ist der Parameter 'table_param' ein Parameter-Array, dessen Inhalt pro Schritt in $table_param aufgenommen wird, falls die durch die beiden Parameter 'condition_type' und 'condition_arg' spezifizierte Bedingung für den Schritt wahr ist. Dabei gibt es für 'condition_type' bisher folgende mögliche Werte:

"parent is": die Bedingung ist für einen Schritt genau dann wahr, falls die Schrittnummer seines (direkten) Parent-Schrittes identisch mit dem Wert des Parameters 'condition_arg' ist, d.h. der Schritt im Subworkflow des Schrittes mit Schrittnummer 'condition_arg' ist, und zwar direkt in der Ebene unter diesem Parent.

"in parentlist": die Bedingung ist für einen Schritt genau dann wahr, falls für einen der indirekten Parent-Schritte des Schrittes dessen Schrittnummer gleich dem Wert des Parameters 'condition_arg' ist, d.h. der Schritt im Subworkflow unter dem Schritt mit Schrittnummer 'condition_arg' ist (evtl. indirekt)

"step in list": die Bedingung ist für einen Schritt genau dann wahr, wenn seine Schrittnummer in der Liste enthalten ist, die der Parameter 'condition_arg' in diesem Fall als Wert hat.

Mit diesen Bedingungen kann man die Schritte eines Subworkflows innerhalb des insgesamt angezeigten Subworkflows (z.B. wenn man alle Schritt eines Workflows auf einmal anzeigt, d.h. $parent="top") oder die Mitglieder-Schritte einer Schleife innerhalb des Subworkflows besonders hervorheben, in dem man entsprechende bedingte Table-Parameter (z.b. für die Anzeige-Farbe) definiert.

Das boolsche Parameter $with_head_and_end gibt an, ob vor den Tabellen-Zeilen der Schritte ein Tabellenkopf und eine Kopfzeile mit 'html_table_head' bzw. dahinter ein Tabellenende mit 'html_table_end' mit erstellt werden soll.

Rückgabe der Funktion ist ein String, der den HTML-Code für die Tabellen der Schrittinformationen enthält

wf_table_edit_frist_datum($parent, $condition_list, $recursive = 0, $table_param = array()) [File 'lib-workflow-table.inc.php']: Diese Funktion ist ein Wrapper für 'wf_table' und erzeugt spezielle eine Tabelle mit Schrittinformationen, die in der Tabellen-Spalte "datumsfeld" ein Formelement zum Ändern des Frist-Datums anbietet (damit wird eine Global-Action um die Änderung dann durchzuführen).

Die Parameter $parent und $recursive wählen dabei die anzuzeigenden Schritte aus wie bei 'wf_table'. Der Parameter $table_param wird direkt an 'wf_table' weitergegeben. Die Liste der bedingten Table-Parameter-Arrays $condition_list wird pro Listenelement um Table-Parameter für das Formular der Datumsänderung erweitert. Dabei reicht es hier,wenn jedes Listenelement als Array nur die Parameter 'condition_type' und 'condition_arg' enthält . (vgl. table oben).

D.h nur bei Schritten, die einer Bedingung in $condition_list entsprechen, wird die Möglichkeit zum Ändern des Frist-Datums angeboten. So kann z.B. der gesamte Workflow angezeigt werden und doch sind nur beispielsweise nur bei Schritten eines bestimmten Subworkflows oder Mitgliedern einer Schleife die Fristen zu ändern.

Um die gesamte per 'wf_table' erstellte Tabelle wird noch ein Formular-Rahmen (Formbeginn - bzw. Ende) gelegt. Als String wird dann der gesamte HTML-Code der Tabelle zurückgeliefert.

3.4 Funktionsweise von 'workflow.php'

Das Hauptprogramm des Workflow-Systems liegt in dem php-Skript 'workflow.php'. Dieses Skript erzeugt bei Aufruf eine dynamische HTML-Seite, die bestimmte Aspekte eines gegebenen Workflows, z.B. die des aktuellen Workflow-Schrittes, darstellt und Links bzw. Formelemente enthält, um das Workflow-System vom Benutzer zu steuern.

Dabei wird das Verhalten - v.a. der Anzeige - des Skriptes durch das beim Aufruf (per Form oder als Teil der URI) übergebene Parameter-Array $param gelenkt. Teile des Arrays können ggf. auch von früheren Aufrufen aus gespeicherten Session-Variablen übernommen werden. Im folgenden wird der Inhalt des Arrays näher beschrieben:

Wichtigster Parameter innerhalb von $param ist 'wf_nr', der die Workflownummer des gewünschten Workflows angibt.

Ein weiterer wichtiger Parameter aus $param ist 'action'. Mögliche Werte dafür sind "Workflow", "Extrapage" und "Extrapage-alone". Die beiden letzteren dienen dabei dem Aufruf einer Extrapage. Im Falle von $param[action]="Workflow" wird dagegen der Inhalt eines bestimmten Workflow-Schrittes angezeigt. Die Schrittnummer dieses Schrittes wird dabei durch den Inhalt von $param[wf_step] festgelegt bzw. falls $param[wf_step] nicht gesetzt ist, wird es mit dem Wert "top" (Schrittnummer des Übersichts-Schrittes) initialisiert. Der Parameter $param[wf_step_position] legt dann fest, wie die Module des durch $param[wf_step] angegebenen Workflow-Schrittes aufgerufen werden und wie ihre Ausgabe in die HTML-Seite eingebettet wird. Mögliche Werte für den Parameter 'wf_step_position' sind hierbei "Main" und "Modulpage". Beim ersteren erfolgt ein normaler Aufruf, d.h. Ausführung und Anzeige des Schrittes mit allen seinen indirekten Parent-Schritten und allen seinen Modulen. Bei "Modulpage" wird einem Modul des Schrittes ermöglicht die HTML-Ausgabe vollständig selbst zu bestimmen.

3.4.1 Standard-Aufruf und Anzeige eines Workflow-Schrittes

Der Standardfall ist $param[wf_step_position]="Main", der auch bei nicht gesetztem $param[wf_step_position] gewählt wird. Hier wird die HTML-Anzeige bzw. der Aufruf aller Module des Schrittes eingebettet in einen vorgegebenen Rahmen. In der Ausgabe erscheint zunächst der Name des Workflows, evtl. Warnungen über überschrittene Zeitfristen.

Darauf werden mit der Funktion 'wf_handle_parent_steps' aus 'lib-workflow-subworkflow.inc.php' der Reihe nach die indirekten Parent-Schritte - beginnend beim top-Schritt - des durch $param[wf_step] gegebenen Schrittes aufgerufen, um eine kurze Anzeige (bzw. evtl. Steuerung) des Zustands des unter ihnen liegenden Subworkflows zu ermöglichen. Es werden hier pro Parent-Schritt nur Module vom Typ "parent" (und nicht Module vom Typ "normal") verwendet, und zwar wird jeweils die Modul-Funktion "head" aufgerufen. Das Array $control_args für diese 'head'-Aufrufe der Parent-Module enthält dabei folgende Parameter:

step_nr: Schrittnummer des jeweiligen Parent-Schrittes

parent_args: Array, das Variablen enthält, die von den indirekten Parents dieses aufgerufenen Parents übergeben werden, beim Aufruf des top-Schrittes ist dieses Array leer.

output_off: boolsches Flag, falls 'true' sollte Ausgabe auf Stdout während des Aufrufes des Parent-Moduls unterlassen werden, um von den Modulen des Schrittes selbst die Ausgabe bestimmen zu lassen. Ist in diesem Fall 'false'.

Die Rückgabe jeder 'head'-Funktion eines Parent-Moduls sollte ein Array sein, das folgende Parameter enthält:

parent_args: Array von Variablenzuweisungen, die von diesem Parent an alle (rekursiven) Kinder (d.h. an darunterliegende Parents des Schrittes und den eigentlichen Schritt selbst) über $control_args[parent_args] (siehe oben) übergeben werden sollen. Das hier zurückgegebene Array wird dabei mit den bereits vorhandenen Variablendefinitionen der höheren Parents vereinigt (die beiden Arrays werden rekursiv "gemerget").

subworkflow_mode_old: boolsches Flag, falls 'true', so wird dem Workflow-System angezeigt, dass hier nicht der aktuelle Durchlauf des unter dem Parent-Schritt liegenden Subworkflows bearbeitet wird, sondern ein ehemaliger, bereits abgeschlossener Durchlauf. Dann wird vom Workflow-System kein Workflow-Status (z.B. Schritt offen oder erledigt, Subworkflow aktiv oder nicht) unter diesem Parent-Modul angezeigt bzw. keine Möglichkeiten zur Änderung des Workflow-Status gegeben.

strict_subworkflow: boolsches Flag, falls 'true', wird von diesem Parent bestimmt, dass Module in Schritten des Subworkflows darunter nur dann aufgerufen werden, d.h. Änderungen nur dann möglich sind, wenn dieser Subworkflow auch aktiv ist.

Erst nach den Parent-Schritten werden der Reihe nach alle Module des gegebenen Schrittes $param[wf_step] mit Modulfunktion "main" aufgerufen. Bei jedem dieser Aufrufe hat das Aufruf-Parameter-Array $control_args folgenden Inhalt:

step_nr: Schrittnummer des Schrittes selbst

step_name: Name des Schrittes, d.h. Wert des Attributes 'name' aus der Relation 'workflow_schritt'.

parents: Liste der Schrittnummern aller indirekten Parents, vom direkten Parent bis zum top-Schritt.

isextrapage: boolsches Flag, das hier stets 'false' ist, um anzuzeigen dass das Modul innerhalb eines Workflow-Schrittes und nicht einer Extrapage aufgerufen wird. Dies bedeutet z.B. auch, dass der Parameter 'step_nr' als Schrittnummer und nicht als Extrapagenummer zu interpretieren ist.

subworkflow_mode_old: boolsches Flag, siehe Rückgabe der 'head'-Funktion eines Parent-Moduls oben.

strict_subworkflow: boolsches Flag, das identisch mit dem 'strict_subworkflow'-Flag des direkten Parent-Schrittes ist (siehe Rückgabe der 'head'-Funktion eines Parent-Moduls oben).

parent_args: Array, das Variablen enthält, die die Parent-Schritte über die 'head'-Funktion an die Module dieses Schrittes übergeben (siehe Rückgabe der 'head'-Funktion eines Parent-Moduls oben).

Der Aufruf der Module des Schrittes erfolgt aber nur, wenn bestimmte Bedingungen erfüllt sind. Damit kann man eine zu frühe Bearbeitung dieser Schritte verhindern:

Hierbei spielt eine Rolle, ob die Abhängigkeiten für einen Schritt selbst strikt sind oder nicht. Die Abhängigkeiten eines Schrittes sind strikt, falls das Attribut 'abhaengigkeiten_sind_strikt' aus der Relation 'workflow_schritt' für diesen Schritt 'true' ist oder falls der Parameter 'abhaengigkeiten_sind_strikt' in der Relation 'workflow_param' vorhanden und gleich 'true' ist. In diesem Fall werden die Module des Schrittes nur aufgerufen, falls der Subworkflow, zu dem dieser Schritt gehört, aktiv ist und alle normalen Abhängigkeiten des Schrittes erfüllt sind.

Weiterhin werden die Schritt-Module dieses Schrittes auch nur dann aufgerufen, wenn bei jedem der Parent-Schritte bereits alle deratigen strikten Abhängigkeiten erfüllt sind. Ist nämlich bei einem der Parents eine solche strikte Abhängigkeit (Abhängigkeiten für den Parent selbst sind strikt und zugleich der Subworkflow in dem dieser sich befindet, nicht aktiv oder existieren unerfüllte, normale Abhängigkeiten für den Parent) nicht erfüllt, so wird bereits dessen 'head'-Funktion und auch die 'head'-Funktion aller darunterliegender Parents nicht mehr ausgeführt.

Jeder Parent-Schritt, d.h. jedes Parent-Modul eines Parents, hat darüberhinaus die Möglichkeit, das Aufrufen von Schritt-Modulen des darunterliegenden Subworkflow zu verbieten, falls der Subworkflow nicht aktiv ist. Dazu kann in der Rückgabe der 'head'-Funktion des Parent-Moduls der Parameter 'strict_subworkflow' auf 'true' gesetzt werden.

Damit werden die Schritt-Module des Schrittes selbst nur dann aufgerufen, wenn eine der folgenden Bedingungen erfüllt ist:

- Einer der Parents setzte das Flag 'subworkflow_mode_old' in seiner Rückgabe auf 'true' und kündigte damit an, dass Module von Schritten des Subworkflows darunter nicht auf Daten des evtl. gerade aktiven Subworkflow-Durchlaufes zugreifen, sondern nur zum Nach-Bearbeiten eines früheren Durchlaufes aufgerufen werden.

- Das Flag 'subworkflow_mode_old' ist 'false' (d.h. normale Bearbeitung des evtl. gerade aktiven Durchlaufes des Subworkflow dieses Schrittes), bei keinem Parent gibt es unerfüllte strikte Abhängigkeiten, der Subworkflow des Schrittes ist nicht aktiv, aber der direkte Parent setzte das Flag 'strict_subworkflow' auf 'false' und für den Schritt selbst sind die eigenen Abhängigkeiten nicht strikt.
- Das Flag 'subworkflow_mode_old' ist 'false' (d.h. normale Bearbeitung des evtl. gerade aktiven Durchlaufes des Subworkflows dieses Schrittes), bei keinem Parent gibt es unerfüllte strikte Abhängigkeiten, der Subworkflow dieses Schrittes ist aktiv und alle normalen Abhängigkeiten dieses Schrittes sind erfüllt oder für den Schritt selbst sind die Abhängigkeiten nicht strikt.

Während seiner Ausführung kann ein Modul praktisch beliebige Dinge tun, vor allem sollte es aber ggf. Datenbank-Operationen durchführen und eine passende Ausgabe auf Stdout erzeugen.

Um innere Abhähngigkeiten (im Schritt selbst) zu realisieren, dient die globale Variable $param[intern_conditions_not_satisfied]. Jedes Modul des Schrittes hat die Möglichkeit den Wert dieser Variable auf 'true' zu setzen, um dem Workflow-System so mitzuteilen, dass in diesem Schritt noch unerfüllte, innere Abhängigkeiten vorhanden sind, und er somit noch nicht als 'erledigt' gekennzeichnet werden kann.

3.4.2 Einzel-Aufruf eines Moduls eines Workflow-Schrittes mit eigener Darstellung

Manchmal ist es erwünscht, dass auf einer HTML-Seite nur ganz bestimmte Informationen auftauchen. Z.B. wenn man den Inhalt der HTML-Seite als Vorlage zum Ausdrucken von Datenbestands-Listen verwenden möchte. Dabei ist es dann ungünstig, wenn z.B. Überschriften wie der Name des Workflows oder des aktuellen Workflow-Schritts, Erklärungen oder Status des Workflows mitangezeigt wird.

Daher besteht die Möglichkeit die vollständige HTML-Ausgabe einer bestimmten Funktion eines Moduls innerhalb eines Workflow-Schrittes zu überlassen. Der Parameter 'wf_step_position' im Array $param erhält dabei statt "Main" den Wert "Modulpage", die Parameter 'wf_step' und 'action' enthalten wie bei normaler Darstellung eines Workflow-Schrittes die Schrittnummer bzw. den Wert "Workflow". Hierbei legen die Parameter 'wf_module_nr' und 'wf_module_funktion' im Array $param die Nummer des Moduls des Schrittes (vgl. Attribut 'modulnummer' in Relation 'workflow_schritt_modul') und die Modulfunktion (im Sinne der Funktion 'wf_call_moduls') fest. Nur das damit identifizierte Modul des Schrittes wird mit der gegebenen Modulfunktion aufgerufen, um die gesamte HTML-Ausgabe (incl. <html>-Tags) zu erzeugen. Ist der Parameter 'wf_module_funktion' gleich dem leeren String, so wird der Defaultwert "fullpage" verwendet. Vor Aufruf des einzelnen Moduls werden noch mittels der Funktion 'wf_handle_parent_steps' aus 'lib-workflow-subworkflow.inc.php' für alle indirekten Parents des aktuellen Schrittes die 'head'-Funktionen ausgeführt. Dabei wird diesen 'head'-Funktionen aber im Array $control_args mit dem Parameter 'output_off' mitgeteilt, keine Ausgabe auf Stdout zu generieren. Der Aufruf der 'head'-Funktionen der Parents dient hierbei nur dazu den Parameter 'parent_args' für den Modulaufruf mit Variablenbelegungen der Parents zu initialisieren. Das Parameter-Array $control_args für den einzelnen Modul-Aufruf ist identisch mit dem für den Aufruf der 'main'-Funktion aus 3.4.1. Das Einzelmodul darf nicht von einem der speziellen Modul-Typen "parent", "end" oder "loop" sein, sondern muss vom Typ "normal" sein. Es wird erwartet, dass die gesamte HTML-Ausgabe während des einzelnen Modulaufrufes geschieht.

3.4.3 Aufruf und Darstellung einer Extrapage

Jeder Workflow kann zusätzlich zu den Workflow-Schritten, bei denen durch ihre Abhängigkeiten untereinander geregelt wird, wann sie bearbeitbar bzw. als 'erledigt' kennzeichenbar sind, sogenannte Extrapages enthalten. Diese haben keinerlei Abhängigkeiten und können zu jeder Zeit im Workflow ausgeführt bzw. angezeigt werden. Anwendung finden sie z.B. bei der Bearbeitung von Stammdaten wie Studenten, Mitarbeitern und Räumen.

Jede solche Extrapage kann dabei wie Workflow-Schritt aus einem oder mehreren Modulen bestehen. Hierbei sind jedoch nur Module vom Typ “normal” zugelassen. Die Definition von Extrapages, sowie ihre Module und deren Parameter wird in eigenen Datenbank-Relationen - getrennt von den Daten der Workflow-Schritte - festgelegt (siehe Relationen ’workflow_extrapage’, ’workflow_extrapage_modul’ und ’workflow_extrapage_modul_param’). Um in einem Modul zwischen einem Aufruf innerhalb eines Workflow-Schrittes und eines Aufrufes innerhalb einer Exrapage zu unterscheiden, wird in $control_args der Parameter ’isextrapage’ auf ’true’ gesetzt.

Um eine Extrapage aufzurufen wird im Array $param der Parameter ’action’ statt “Workflow” auf “Extrapage” bzw. “Extrapage-alone” gesetzt. “Extrapage-alone” unterscheidet sich dabei von “Extrapage” nur dadurch, dass keine Navigations-Verweise zur Übersichtsseite des Workflows angezeigt werden. Dies ist sinnvoll, falls die Extrapages eines Workflows von außerhalb des Workflows - praktisch unabhängig vom Workflow selbst - aufgerufen werden sollen. Insbesondere könnte ein Dummy-Workflow in der Datenbank angelegt werden, der nur Extrapages zur Verwaltung der Stammdaten und keine echten Workflow-Schritte enthält.

Die Nummer der Extrapage ist im Parameter ’extrapage_nr’ im Array $param enthalten (ähnlich wie ’wf_step_nr’ bei ’action’=”Workflow”). Ebenso existiert ein Parameter ’extrapage_position’, der als mögliche Werte “Main” oder “Modulpage” hat (ähnlich wie ’wf_step_position’ bei ’action’=”Workflow”).

Hat dieser Parameter den Wert “Main”, so wird ein minimaler HTML-Rahmen vor Aufruf der Extrapage-Module ausgegeben. Dieser enthält z.B. den Namen der Extrapage sowie Verweise zu den anderen Extrapages und ggf. einen Verweis zum top-Schritt des Workflows. Jedes Modul der Extrapage wird dann mit Modulfunktion “main” aufgerufen, wobei das Array $control_args folgendes Aussehen hat:

isextrapage: boolsches Flag, hier stets gleich ’true’, zeigt an, dass das Modul in einer Extrapage aufgerufen wird

step_nr: Nummer der Extrapage, d.h. der Wert von $param[extrapage_nr]

step_name: Name der Extrapage, d.h. das Attribut ’name’ aus der Relation ’workflow_extrapage’

Bei $param[extrapage_position]=“Modulpage” wird ähnlich wie bei $param[action]=”Workflow” einer gegebenen Modulfunktion eines bestimmten Moduls der Extrapage ermöglicht die gesamte HTML-Ausgabe zu erzeugen. Die Modulnummer und die Modulfunktion werden hier durch die Parameter ’extrapage_modul_nr’ und ’extrapage_module_funktion’ aus $param (analog ’wf_module_nr’ und ’wf_module_funktion’ bei ’action’=”Workflow”) bestimmt. Das einzeln aufgerufene Modul erhält in $control_args die gleichen Parameter wie bei $param[wf_step_position]=”Main” (siehe oben).

3.5 Aufbau der verschiedenen Modultypen

Hier soll für jeden der verschiedenen Modultypen beschrieben werden, wie ein Modul aufgebaut ist.

Grundsätzlich besteht ein Modul vom typ <typ> (’parent’, ’end’, ’loop’ oder ’normal’) mit Modul-Namen <modulname> aus einer oder mehreren php-Funktionen jeweils mit php-Funktions-Namen “<typ>_module_<modulname>_<modulfunktion>” (bei Typ “parent”, “end” oder “loop”) bzw. “module_<modulename>_<modulfunktion>” (bei Typ “normal”) (vgl. Funktion ’wf_call_moduls’ in 3.3.1), wobei <modulfunktion> die von der php-Funktion realisierte Modulfunktion (im Sinne von ’wf_call_moduls’) ist. Jedes Modul verfügt über die Modulfunktion “main”, d.h. eine php-Funktion mit Namen “<typ>_module_<modulename>_main” bzw. “module_<modulename>_main” (Typ ’normal’), die i.a. ausgeführt wird, wenn das Modul als Teil des aktuellen Workflow-Schrittes aufgerufen wird.

Als Parameter erhält eine solche Modulfunktion beim Aufruf durch ’wf_call_moduls’ zwei Parameterarrays mit Namen $control_args und $module_args. Das zweite Array besteht dabei genau aus den Modul-Parametern aus der Relation ’workflow_modul_param’. Diese sind während eines

Workflows nicht veränderbar und können pro Modul eines Schrittes und pro Schritt unterschiedlich sein. Der Inhalt von $control_args dagegen wird vom Aufrufer der Funktion 'wf_call_moduls' bestimmt. In diesem Array befinden sich i.a. Variablen, die den aktuellen Workflow-Zustand in einer bestimmten Weise beschreiben. Ihr Inhalt sollte jeweils zur aufgerufenen Modulfunktion passen. (siehe Funktion 'wf_call_moduls' in 3.3.1)

3.5.1 Module vom Typ 'normal'

Ein Modul vom Typ 'normal' beinhaltet keinerlei Workflow-Steuerungsfunktionalität. Es wird z.B. zur Verwaltung von Anwendungsdaten (z.B. Räume, Studenten einer Vorlesung) oder zum Ausführen bestimmter Aktionen bzgl. der eigentlichen Anwendung (z.B. Übungs-Scheine erstellen, Mail verschicken) verwendet.

Ein solches Modul muss nur eine "main"-Modulfunktion enthalten. Hierbei wird unterschieden, ob das Modul für einen Workflow-Schritt oder eine Extrapage aufgerufen wird (boolscher Parameter 'isextrapage' in $control_args).

Falls es in einem Workflow-Schritt aufgerufen wird, so kann es im Array $control_args auf folgende Parameter zugreifen (vgl. 3.4.1):

step_nr: Schrittnummer des Schrittes, für den das Modul aufgerufen wird. Bei Schritten vom Typ "normal" z.B. interessant, um beim Aufruf der Funktion 'wf_temp_data_get' (siehe oben 3.3.5) die eigenen temporären Daten zu finden oder auch um eindeutige Bezeichner in eigenen Form-Variablen zu wählen .

step_name: Name des Schrittes, d.h. Wert des Attributes 'name' aus der Relation 'workflow_schritt', für den das Modul aufgerufen wird

parents: Liste der Schrittnummern aller indirekten Parents des Schrittes, für den das Modul aufgerufen wird, vom direkten Parent bis zum top-Schritt.

isextrapage: boolsches Flag, das hier stets 'false' ist, um anzuzeigen, dass das Modul innerhalb eines Workflow-Schrittes und nicht einer Extrapage aufgerufen wird. Dies bedeutet z.B. auch, dass der Parameter 'step_nr' als Schrittnummer und nicht als Extrapagenummer zu interpretieren ist.

subworkflow_mode_old: boolsches Flag, das anzeigt, ob tatsächlich Daten des derzeit aktuellen Subworkflow-Durchlaufes bearbeitet werden ('false') oder Daten eines früheren Durchlaufes ('true'). Für Module vom Typ "normal" nicht wesentlich.

strict_subworkflow: boolsches Flag, das identisch mit dem 'strict_subworkflow'-Flag des direkten Parent-Schrittes ist (siehe Rückgabe der 'head'-Funktion eines Parent-Moduls in 3.4.1). Für Module vom Typ "normal" unwichtig.

parent_args: Array, das Variablen enthält, die die Parent-Schritte über die 'head'-Funktion an die Module dieses Schrittes übergeben (siehe Rückgabe der 'head'-Funktion eines Parent-Moduls in 3.4.1). Hier werden die für einen bestimmten Subworkflow-Durchlauf relevanten Daten, wie z.B. die Nummer der aktuellen Klausur, übergeben.

Spezielle Rückgaben sind nicht zu machen.

Prinzipiell kann jedes Modul, wenn es für einem Workflow-Schritt aufgerufen wird, jede in 3.3.2 bis 3.3.8 beschriebene Funktion aufrufen, die Informationen über den Status ihres Schrittes ermittelt. Bei Modulen vom Typ "normal" ist dies aber i.a. nicht notwendig, da sie sich nicht mit der Workflow-Steuerung befassen. Aber die Funktionen zum Zugriff auf temporäre Daten, wie z.B. 'wf_temp_data_get', 'wf_temp_data_set', sind auch dafür gedacht von normalen Modulen benutzt zu werden.

Wird ein Modul vom Typ 'normal' jedoch für eine Extrapage aufgerufen($control_args[isextrapage]='true'), so enthält $control_args nur die Parameter 'isextrapage' (hier gleich 'true'), step_nr (ist hier als Extrapagenummer aufzufassen) und step_name (als Name der Extrapage) (vgl. 3.4.3). Auch sollte dann nicht versucht werden Workflow-Status des Schrittes $control_args[step_nr] zu ermitteln, da der Parameter 'step_nr' eine Extrapagenummer und nicht die Schrittnummer eines Workflow-Schrittes bezeichnet.

Darüberhinaus kann ein Modul vom Typ 'normal', wie auch jedes Modul anderen Typs, weitere Modulfunktionen enthalten, die die gesamte HTML-Seite selbst erzeugen, um mit $param[wf_step_position]="Modulepage" allein auf einer HTML-Seite dargestellt zu werden (siehe 3.4.2).

3.5.2 Module vom Typ 'parent'

Die Module vom Typ 'parent', oder kurz Parent-Module, kommen nur in Workflow-Schritten vom Typ 'parent' vor. Sie dienen dort dazu, die Steuerung über den Subworkflow unter dem Parent-Schritt zu übernehmen.

Ein Parent-Modul muss wie ein normales Modul auch die Modulfunktion "main" besitzen. Der Inhalt ihrer $control_args ist derselbe wie bei normalen Modulen und es gibt keine Rückgaben. Diese wird aufgerufen, falls der Parent-Schritt selbst der aktuell angezeigte und verarbeitete Workflow-Schritt ist, d.h. seine Schrittnummer der Wert von $param[wf_step] ist (siehe 3.4). Im Gegensatz zu normalen Modulen wird aber das Parent-Modul i.a. Funktionen zum Lesen und Verändern des Status des Subworkflows verwenden. Typische Funktionen zum Status-Lesen hierbei sind 'wf_step_status', 'wf_subworkflow_status', 'wf_step_durchlaufnummer'. Zum Aktivieren bzw. Deaktivieren des Subworkflows könnten die Funktionen 'wf_subworkflow_open' bzw. 'wf_subworkflow_close' benutzt werden. Jedoch ist die indirekte Aufruf-Variante mit Global-Actions (z.B. via 'wf_globalaction_form_element' siehe 3.3.6) vorzuziehen.

Das Schließen des Subworkflows unter dem Parent-Schritt sollte jedoch i.a. dem Subworkflow selbst, d.h. den End_Schritten des Subworkflows vorbehalten bleiben, so dass sich der Parent-Schritt meist nur um die Aktivierung kümmern muss. Jedoch kann der Parent-Schritt über die Funktion 'wf_notify' (siehe 3.3.6) abfragen, ob eine Schließung des Subworkflows im aktuellen php-Skript-Aufruf vor Ausführung des Parent-Moduls erfolgte (z.B. durch Global-Action-Mechanismus). Der Funktionsaufruf von 'wf_notify($parentstep)' gibt ein Array zurück, in dem der Parameter 'subworkflow_has_closed' gleich '1' ist, falls eine Schließung des Subworkflows in diesem Aufruf erfolgte. Ebenso ist der Parameter 'subworkflow_has_opened' gleich '1', falls eine Aktivierung des Subworkflows in diesem Aufruf erfolgte.

Weiterhin verwalten Parent-Module oft zusätzliche temporäre Daten (meist gespeichert in der Relation 'workflow_temp_data'), die für den aktuellen Subworkflow-Durchlauf relevant sind, d.h. Daten die Verbindung zwischen den Workflow-Status-Informationen und den Anwendungsdaten herstellen. Ein Beispiel hierfür wäre die Nummer einer Klausur, die in einem bestimmten Subworkflow-Durchlauf durchgeführt wird. Einerseit ist diese Nummer ein Attribut in den Anwendungsdaten über die Klausuren, andererseits ist sie mit dem aktuellen Subworkflow-Durchlauf verbunden. Es steht den Parent-Modulen jedoch frei, ob sie diese Daten in der Relation 'workflow_temp_data' über die in 3.3.5 beschriebenen Funktionen ('wf_temp_data...') oder in den Anwendungs-Relationen (bei passendem Anwendungs-Datenbank-Schema) selbst speichern.

Um einerseits den Schritten eines Subworkflows diese zusätzlichen temporären Informationen zugänglich zu machen und andererseits kurze Informationen über den aktuellen Subworkflow-Status anzuzeigen, hat jedes Parent-Module eine Modulefunktion mit Namen "head". Bevor die Module des aktuellen Schrittes ausgeführt und angezeigt werden, werden rekursiv von allen indirekten Parent-Schritten die 'head'-Funktion in den Parent-Module aufgerufen (vgl. 3.4.1). Damit hat der Benutzer auch durch die Anzeige einen groben Überblick über die Subworkflow-Struktur. Vor allem aber dienen die head-Funktionen der Übergabe der temporären Informationen an die Schritte des Subworkflows, sowie einiger Subworkflow-Status-Information, die nur der Parent-Schritt kennt, an das globale Workflow-System. Der 'head'-Modulfunktion steht es dabei frei auch Änderungsmöglichkeiten (wie in der "main"-Modulfunktion) dem Benutzer anzubieten.

Das Array $control_args für den Aufruf der 'head'-Funktion eines Parent-Moduls eines Parent-Schrittes hat folgenden Inhalt (vgl 3.4.1):

step_nr: Schrittnummer des jeweiligen Parent-Schrittes

parent_args: Array, das die temporären Variablen enthält, die von den indirekten Parents dieses aufgerufenen Parents übergeben werden. Beim Aufruf des top-Schrittes ist dieses Array leer. Der aktuelle Schritt erhält schließlich alle diese Parameter gesammelt von allen seinen indirekten Parents.

output_off: boolsches Flag, falls 'true' sollte Ausgabe auf Stdout während des Aufrufes des Parent-Moduls unterlassen werden, um von den Modulen des Schrittes selbst die HTML-Ausgabe bestimmen zu lassen ($param[wf_step_position]="Modulpage, vgl 3.4.2).

Die Rückgabe jeder 'head'-Funktion eines Parent-Moduls sollte ein Array sein, das folgende Parameter enthält:

parent_args: Array von Variablenzuweisungen, die von diesem Parent an alle (rekursiven) Kinder (d.h. an darunterliegende Parents des Schrittes und den eigentlichen Schritt selbst) über $control_args[parent_args] (siehe oben) übergeben werden. Das hier zurückgegebene Array wird dabei mit den bereits vorhandenen Variablendefinitionen der höheren indirekten Parents vereinigt.

subworkflow_mode_old: boolsches Flag, falls 'true', so wird dem Workflow-System angezeigt, dass hier nicht der aktuelle Durchlauf des unter dem Parent-Schritt liegenden Subworkflows bearbeitet wird, sondern ein ehemaliger, bereits abgeschlossener Durchlauf. Dann wird vom Workflow-System kein Workflow-Status (z.B. Schritt offen oder erledigt, Subworkflow aktiv oder nicht) unter diesem Parent-Modul angezeigt bzw. keine Möglichkeiten zur Änderung des Workflow-Status gegeben.

strict_subworkflow: boolsches Flag, falls 'true', wird von diesem Parent bestimmt, dass Module in Schritten des Subworkflows darunter nur dann aufgerufen werden, d.h. Änderungen nur dann möglich sind, wenn dieser Subworkflow auch aktiv ist.

Die zurückgegebenen Parameter 'subworkflow_mode_old' und 'strict_subworkflow' haben die in 3.4.1 beschriebene Wirkung.

Wie normale Module können auch Parent-Module weitere Modulfunktionen enthalten, die aufgerufen bei $param[wf_step_position]="Modulepage" (siehe 3.4.2) selbst eine ganze HTML-Seite allein erzeugen.

3.5.3 Module vom Typ 'end'

Nur in End-Schritten kommen Module vom Typ 'end' (kurz End-Module) vor. Sie dienen der Deaktivierung ihres Subworkflows. In aller Regel sollten sie erst dann eine Deaktivierung zulassen, wenn ihre normalen Abhängigkeiten es erlauben.

End-Module müssen nur die Modulfunktion "main" (gleiche $control_args wie beim Typ "normal", keine Rückgaben) enthalten. Darin können sie, v.a. wenn alle ihre normalen Abhängigkeiten erfüllt sind, es dem Benutzer gestatten ihren Subworkflow zu deaktivieren. Die Deaktivierung wird am besten indirekt mit Global-Actions (z.b. über die Funktion 'wf_globalaction_form_element', siehe 3.3.6) ausgelöst. Denn wenn ein End-Modul eines End-Schrittes seinen Subworkflow selbst über die Funktion 'wf_subworkflow_close' sperrt, werden z.B. in der HTML-Anzeige über dem End-Schritt von den 'head'-Funktionen der indirekten Parent-Schritte noch veraltete Informationen (über noch nicht geschlossenen Subworkflow) angezeigt. Der Status des Subworkflows und des End-Schrittes selbst kann mit Funktionen wie 'wf_subworkflow_status', 'wf_step_status', etc.

abgefragt werden. Insbesondere sollte auf den Parameter $control_args[subworkflow_mode_old] geachtet werden. Ist dieser glich 'true', sollte das Modul am besten gar nichts tun, da es sich hier um die Bearbeitung eines abgeschlossenen Subworkflow-Durchlaufes handelt.

End-Module können wie alle Module bei Bedarf über die 'wf_temp_data...'-Funktionen (siehe 3.3.5) zusätzliche, temporäre Informationen verwalten. Sie können auch weitere Modulfunktionen zum Aufbau ganzer HTML-Seiten durch eine einzige Modulfunktion ($param[wf_step_position]="Modulpage", siehe 3.4.2) enthalten.

Speziell ist zu End-Schritten zu sagen, dass das Workflow-System nie eine Änderung des 'erledigt'-Status eines End-Schrittes anbietet. D.h. wenn der Subworkflow eines End-Schrittes aktiv ist, so hat dieser stets den 'erledigt'-Status 'offen'.

3.5.4 Module vom Typ 'loop'

Loop-Schritte enthalten Module vom Typ "loop" (kurz Loop-Modul), die die Steuerung einer Schleife innerhalb eines Subworkflows verrichten.

Loop-Module haben die Modulfunktion "main" (gleiche $control_args wie bei normalen Modulen, keine Rückgaben), die normalerweise aufgerufen wird, wenn der Loop-Schritt der aktuelle Workflow-Schritt ist. Status des Loop-Schrittes und der Schleife kann mit Funktionen wie 'wf_step_status', 'wf_step_active', 'wf_steps_in_loop', etc. abgefragt werden. Ein neuer Durchlauf der Schleife, d.h. das Zurücksetzen des 'erledigt'-Status aller Mitglieder der Schleife des Loop-Schrittes auf 'offen', sollte mittels Global-Action (z.B. über die Funktion 'wf_globalaction_form_element' in einem Formular, siehe 3.3.6) angestoßen werden, um inkonsistente Anzeigen über 'offene' und 'erledige' Schritte auszuschließen. Andernfalls kann die Aktion auch direkt über 'wf_loop_iterate($loopstep)' angestoßen werden. Weiter kann ein Loop-Modul z.B. Links zu den Mitglieder-Schritten seiner Schleife oder den Status der Mitglieder-Schritte anzeigen. Auch Loop-Schritte sollten darauf achten, ob der Parameter $control_args[subworkflow_mode_old] gleich 'true' ist oder nicht. Falls dieser gleich 'true' ist, wird ein früherer Durchlauf des Subworkflow des Loop-Schrittes bearbeitet und der Loop-Schritt sollte keine Änderung, d.h. keinen neuen Schleifen-Durchlauf, erlauben.

Weiterhin sollte ein Loop-Modul noch eine Modulfunktion "main_strict_deps" haben, die die gleichen $control_args wie die Modulfunktion "main" erwartet. Diese wird statt "main" für alle Loop-Module eines Schrittes aufgerufen, falls der Schritt aufgrund strikter Abhängigkeiten (siehe 3.4.1) als nicht bearbeitbar erkannt wird. Hierin kann das Loop-Modul den Zustand der Schleife und evtl. Links zu Mitglieder-Schritten der Schleife auch unter diesen verschärften Bedingungen nach eigenem Belieben anbieten.

Auch Loop-Schritte können eigene, temporäre Daten über die 'wf_temp_data...'-Funktionen verwalten. Sie können auch weitere Modulfunktionen zum Aufbau ganzer HTML-Seiten durch diese einzige Modulfunktion ($param[wf_step_position]="Modulpage", siehe 3.4.2) enthalten.

3.6 Anlegen eines neuen Workflows

Zum Festlegen eines neuen Workflows dient eine XML-Konfigurationsdatei. In dieser wird der Aufbau des neuen Workflows hierarchisch beschrieben. Mit dem Perl-Skript "wfcreate.pl" werden aus diesen XML-Daten SQL-INSERT-Anweisungen zum Anlegen bzw. Abspeichern der Informationen des neuen Workflows in der Datenbank erstellt. Die XML-Daten haben folgendes Format:

Ein XML-Element entspricht hierbei i.d.R. einem Eintrag in eine DB-Relation. Die XML-Attributswerte werden dabei zu den Attributswerten. Die Namen der Elemente und Relationen bzw Attribute sind jeweils allerdings unterschiedlich, um dem Benutzer anschaulichere Namen zu erlauben bzw. um die gleiche Bezeichnung für verschiedenartige Elemente zu erlauben, je nachdem wo sie in der Hierarchie stehen:

Die XML-Element- bzw. Attributsnamen werden wie folgt in DB-Relationsnamen bzw. Attributsnamen umgesetzt: [XML-Attribute in eckigen Klammern müssen nicht spezifiziert werden und werden automatisch ergänzt]

workflow-Element, das das Top-Element in der XML-Struktur bildet:

XML-Element:	workflow	DB-Relation:	workflow
XML-Attribute:	nummer	DB-Attribute:	workflownummer
	name		name
	bemerkung		bemerkung

"nummer" ist hierbei die Nummer, die den neuen Workflow eindeutig identifiziert. Sie taucht z.B. auch als Parameter wf_nr beim Aufruf von workflow.php auf.

param-Element als Kind des workflow-Elementes:

XML-Element:	param [workflow]	DB-Relation	workflow_param
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	[nummer]		paramnr
	name		paramkey
	value		paramvalue

Hiermit sind globale, statische Workflow-Parameter festlegbar. Der Wert "value" steht später jedem Modul des Workflows als php-Variable $wf_global["name"] zur Verfügung. Die Nummer wird, falls nicht spezifiziert einfach pro Parameter hochgezählt und dient nur in der Datenbank als Teil-Schlüssel (da "name" länger als 255 Zeichen sein kann, kam es statt dessen als Teilschlüssel nicht in Frage).

Der besondere Parameter mit name="abhaengigkeiten_sind_strikt" spezifiziert, falls mit sein value mit "1" belegt ist, dass die Abhängigkeiten von allen strikt sind, d.h. Schritte erst bearbeitbar sind, wenn ihre Abhängigkeiten erfüllt sind. Ist value="0", so kann dies bei jedem Schritt einzeln entschieden werden.

schritt-Element (i.d.R. als Kind vom workflow-Element oder eines subworkflow-Elementes):

XML-Element:	schritt	DB-Relation:	workflow_schritt
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	nummer		schrittnummer
	name		name
	typ/[typ2]		typ
	[vorgaenger]		voriger_schritt
	[nachfolger]		naechster_schritt
	[sub_anfang]		erster_subschritt
	[sub_ende]		letzter_subschritt
	[vater]		parent_schritt
	erklaerung		erklaerung
	abhaengigkeiten_sind_strikt		abhaengigkeiten_sind_strikt
	status		status
	subworkflow_aktiv		subworkflow_aktiv
	durchlaufnummer		durchlaufnummer
	erledigt_datum		erledigt_datum
	frist_datum		frist_datum
	bemerkung		bemerkung
	subworkflow_nummer_prefix		

Hiermit sind die Schritte des neuen Workflows definierbar. Das Attribut "nummer" ist ein (kurzer) String, der den Schritt eindeutig identifiziert. Es wurde String, und nicht etwa Integer, als Datentyp gewählt, um hierarchische Nummerierung wie z.B. 2.5.1 (für Subworkflow-Hierarchien) zu ermöglichen.

Die Nummern können weitgehend automatisch (in hierarchischer Form) generiert werden. Es sollte jedoch stets einen obersten top-Schritt geben, dessen Nummer im Konfigfile angegeben werden muss. Dieser oberste Schritt dient einerseits der Strukturierung und seine Schrittseite dient der Übersicht über den gesamten Workflow. Damit “workflow.php” später diesen Schritt kennt, sollte er im Moment stets den Namen “top” erhalten.

Kinder-Schritte innerhalb eines Subworkflows erhalten bei Nicht-Angabe der Nummer automatisch die Nummer des Parent-Schrittes konkateniert mit einem “.” und einer Nummer entsprechend ihrer Reihenfolge im Subworkflow. Es besteht aber mit dem speziellen XML-Attribut “subworkflow_nummer_prefix” im XML-Element des Parent-Knotens die Möglichkeit, ein anderes Präfix statt der Nummer des Parents verknüpft mit dem String “.” für den automatischen Schrittnamen zu wählen. Z.B. wird man i.d.R. im top-Schritt subworkflow_nummer_prefix=”” (leerer String) benutzen, um auf der eigentlich obersten Workflow-Ebene nur einfache Nummern, wie z.B. “1”, für die Schritte und nicht etwa “top.1” zu erhalten.
Die automatische Nummerierung der Schritte bereitet allerdings ein Problem: Schrittnummern identifizieren einen Schritt, z.B. werden beim Definieren einer Abhängigkeit zwischen zwei Schritten die Schrittnummern angegeben. Da also beim Erstellen der Konfiguration für das Festlegen von z.B. Abhängigkeiten die Schrittnummern benötigt werden, gibt es ein Konzept, um einem bestimmten Attribut eines bestimmten Elementes im XML-Baum, dessen Wert noch nicht bekannt ist (da er erst später vom Erstell-Skript automatisch hinzugefügt wird), einen globalen Alias-Namen (unabhängig vom gegebenen Element bzw. Attribut) zuzuordnen. Statt den Wert des XML-Attribut des jeweiligen XML-Elements explizit anzugeben, gibt man (im XML-Element) statt dessen ein XML-Attribut mit Namen “alias:<aliasname>” und dem Name des zu repräsentierenden Attributs als Wert an. Z.B. definiert alias:anmeldung=”nummer” als Attribut in einem bestimmten Schritt-Element den Alias “anmeldung” für das (zukünftige) Attribut “nummer” des bestimmten Schritt-Elementes. Für ein gewöhnliches XML-Attribut, in dessen Wert ein solcher Alias vorkommen und (später) ersetzt werden soll, wird statt dem normalen Namen des XML-Attributs der Name “aliased:<gewöhnlicher Attributname>” verwendet und im Wert des Attributs kann dann ein Alias mit “@(<aliasname>)” referenziert werden. Z.B. würde für die XML-Attribut-Zuweisung aliased:schritt=”@(anmeldung)” innerhalb eines beliebigen Abhängigkeits-Elements irgendwo innerhalb des XML-Baumes (falls der Alias wie im obigen Beispiel definiert wäre) der String “@(anmeldung)” im Attribut “schritt” der Abhängigkeit durch das später erstellte Attribut “nummer” des Schritt-Elements aus dem vorigen Beispiel ersetzt werden. So kann man Abhängigkeiten von Schritten im XML-File angeben, ohne die später automatisch errechnete Nummer des Schrittes zu kennen.

“name” ist ein String, der als kurzer Titel für den Schritt steht.

Das Attribut “typ” hat als mögliche Werte “normal”, “parent” (Parent-Schritt eines Subworkflows), “end” (Subworkflow-Ende) oder “loop” (Schleifenbildungs-Schritt). Bei einem Schritt, der ein subworkflow-Element (s.u.) als Kind hat, bei einem Schritt, der ein workflow_schleifenstart-Element als Kind hat oder einem normalen Schritt muss das Attribut nicht mitangegeben werden. Somit ist i.a. nur ein Schritt mit typ=”end” als Subworkflow-Ende zu kennzeichnen.

Das Attribut “typ2” wird nur intern verwendet, um “typ” -Werte auf die Attributs-Werte in der Datenbank abzubilden.

Die (Darstellungs-)Reihenfolge der Schrittseiten (die “Navigierungs-Reihenfolge “ der Schrittseiten als Web-Seiten), die völlig unabhängig von den (normalen) Abhängigkeiten zwischen den Schritten ist, wird durch die Attribute “vorgaenger” und “nachfolger” beschrieben. Diese werden i.a. aber automatisch durch die Nummern der Schritte, die im schritt-Element vor bzw. nach diesem schritt-Element definiert sind, ersetzt, falls nicht angegeben.

Bei einem Parent-Schritt eines Subworkflows werden mit den Attributen “sub_anfang” und “sub_ende” der Navigations-Beginn (1. Seite) bzw. das Navigations-Ende (letzte Seite) eines Subworkflows spezifiziert. Auch sie müssen nicht mit angegeben werden, sondern werden aus der Verschachtelungsstruktur des subworkflow-Elementes unter dem schritt-Element, das als Container für die schritt-Elemente des Subworkflows dient, bestimmt.

Der Parent-Schritt des Subworkflows, zu dem der aktuell beschriebene Schritt gehört, wird durch das Attribut "vater" beschrieben und kann auch aus der Struktur automatisch erkannt werden.

"erklaerung" ist ein String, der auf der jeweiligen Schrittseite mit angezeigt wird, um dem Benutzer eine kurze Beschreibung des Schrittes zu geben.

Um festzulegen, ob die Abhängigkeiten des Schrittes strikt sein sollen, d.h. der Schritt erst bearbeitbar (nicht nur erledigbar) sein soll, wenn alle Abhängigkeiten erfüllt sind, oder nicht , enthält das Attribut "abhaengigkeiten_sind_strikt" entsprechend "1" oder "0" als boolschen Wert.

"status" hat als mögliche Werte "offen" oder "erledigt". Beim Anlegen eines Workflows ist i.a. nur "offen" sinnvoll, was auch bei Nicht-Angabe dieses Attributs als default genommen wird.

Bei Parent-Schritten gibt "subworkflow_aktiv" als boolscher Wert ("0" oder "1") an, ob der Subworkflow unter dem Parent-Schritt aktiv ist. Auch hier ist meist nur "0" sinnvoll, was auch der Standardwert ist.

Ähnliches gilt für "durchlaufnummer", das die Anzahl Durchläufe eines Subworkflows unter einem Parent-Schritt oder einer Schleife eines Schleifenbildungsschrittes zählt. Es hat als default-Wert "0".

Auch das "erledigt_datum" muss nicht spezifiziert sein, da i.a bei einem neuen Workflow noch kein Schritt erledigt ist. Das "frist_datum" dagegen kann , vor allem für Schritte direkt unterhalb vom top-Schritt, falls gewünscht als SQL-Timestamp (z.B. 2001-09-08), angegeben werden.

Das Attribut "bemerkung" steht für den Wert der Bemerkung (eine Art Notiz-Funktion) zu einem Schritt. Dieser wird später stets auf der Schrittseite angezeigt und kann geändert jederzeit werden. Bei Bedarf kann hiermit ein String für den initialen Wert angegeben werden.

Das subworkflow-Element dient nur der Strukturierung im XML-File und hat selbst keine Entsprechung in der Datenbank (zumindest nicht als ein einzelner Tupeleintrag innerhalb einer Tabelle). Es kennzeichnet die Zugehörigkeit der in ihm enthaltenen Schritte zu einem Subworkflow. Es selbst ist Kind des Schritt-Elements des Parent-Schrittes. Da die Implementierung nicht mehrere, verschiedene Subworkflows unter einem Parent-Schritt ermöglicht, hätte man es auch in der Definition des Konfigformats weglassen können.

modul-Elemente als Kinder eines Schritt-Elementes:

XML-Element:	modul [schritt]	DB-Relation:	workflow_schritt_modul
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	[schritt]		schrittnummer
	[nummer]		modulnummer
	name		modulname

Die einzelnen Module eines Schrittes werden hiermit definiert. "name" ist ein String der Form "<modultyp>:<modulname>". Mögliche Modultypen sind "normal", "parent", "end" und "loop". Der Typ "normal" kann auch weggelassen werden, d.h. dann "name"="<modulname>". "<modulname>" gibt den eigentlichen Namen des Moduls an. Dabei ist ein Modul mit Typ "normal" in der Datei "module-<name>.inc.php definiert, ein Parent-Modul (Typ "parent") in der Datei "parent_module-<modulname>.inc.php", ein Subworkflow-Ende-Modul (Typ "end") in der Datei "end_module-<modulname>.inc.php" und ein Schleifenbildungsmodul (Typ "loop") in der Datei "loop_module-<modulname>.inc.php" definiert.

Hierbei sollten Module vom Typ "parent" sind nur in Schritten vom Typ "parent", Module vom Typ "end" nur in Schritten vom Typ "end" und Module vom Typ "loop" nur in Schritten vom Typ "loop" verwendet werden. Module vom Typ "normal" können überall eingesetzt werden.

Das Attribut "nummer" wird, falls nicht angegeben, einfach der Reihenfolge der Elemente nach durchnummeriert, wobei ein Modul mit kleinerer Nummer vor Modul einem Modul mit größerer Nummer auf einer Schrittseite aufgerufen wird, d.h. bei Nicht-Angabe von "nummer" entscheidet die Reihenfolge der Module in der Konfiguration die Reihenfolge der Module später beim Aufruf einer Schrittseite.

param-Elemente als Kinder eines modul-Elements, das Kind eines Schritt-Elementes ist:

XML-Element:	param [modul.schritt]	DB-Relation:	workflow_schritt_modul_param
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	[schritt]		schrittnummer
	[modul]		modulnummer
	[nummer]		paramnr
	name		paramkey
	value		paramvalue

Ähnlich wie globale Workflow-Parameter lassen sich die Parameter eines Modulschrittes festlegen. Sie stehen dem jeweiligen Modul eines Schrittes später beim Aufruf des Modules i.a. in der Form $module_args[“name”] zur Verfügung.

extrapage-Elemente (i.d.R. Kind vom workflow-Element):

XML-Element:	extrapage	DB-Relation:	workflow_extrapage
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	extrapage		extrapagenummer
	name		name
	flags		flags

Extrapages sind Seiten außerhalb des eigentlichen Workflows und dienen allgemeinen Verwaltungszwecken, die zu jeder Zeit während des Workflow-Ablaufes ausführbar sein sollen, wie z.B. das Eingeben von Basisdaten wie Studenten, Mitarbeitern und Räumen. Das Attribut “extrapage” dient der Identifikation ähnlich wie “nummer” bei Schrittseiten. Wird es nicht spezifiziert , so wird einfach der Reihenfolge nach durchnummeriert. “name” ist wie bei Schrittseiten ein String, der als Titel für die Extrapage dient.

Mit dem Attribut “flags” ist festlegbar, auf welchen Schrittseiten Verweise auf die neue Extraseite vorhanden sind. Es wird hierbei zwischen der Übersichtsseite des Workflows (top-Schritt) und sonstigen Schrittseiten, sowie Seiten-Kopf und Seiten-Ende unterschieden, z.B. erscheint bei “flags=11” ein Verweis zur Extrapage im Kopf und am Ende der Übersichtsseite, sowie am Ende aller sonstigen Schrittseiten:

Bitnummer von flags:	Verweis zur Extrapage:
0	Verweis erscheint im Kopf der Übersichtsseite
1	Verweis erscheint am Ende der Übersichtsseite
2	Verweis erscheint im Kopf aller Schrittseiten
3	Verweis erscheint am Ende aller Schrittseiten

modul-Elemente mit einem exrapage-Element als Vater:

XML-Element:	modul [extrapage]	DB-Relation:	workflow_extrapage_modul
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	[extrapage]		extrapagenummer
	[nummer]		modulnummer
	name		modulname

Extrapages sind genauso wie Schrittseiten aus Modulen zusammengesetzt. Allerdings können nur Modul vom Typ “normal” verwendet werden. Um im Modul unterscheiden zu können (falls notwendig), ob es innerhalb einer Extrapage oder einer Workflow-Schrittseite aufgerufen wird gibt es die Variable $is_extrapage.Ergibt ihr boolscher Wert “wahr”, so ist das Modul innerhalb einer Extrapage aufgerufen worden.

param-Elemente in einem modul-Element in einem extrapage-Element:

XML-Element:	param [modul.extrapage]	DB-Relation:	workflow_extrapage_modul_param
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	[extrapage]		extrapagenummer
	[modul]		modulnummer
	[nummer]		paramnr
	name		paramkey
	value		paramvalue

Analog wie bei Modulen eines Schrittes hat kann ein Modul für eine Extrapage Parameter haben.

abhaengigkeits-Elemente (i.d.R. als Kinder eines Schritt-Elements):

XML-Element:	abhaengigkeit	DB-Relation:	workflow_schritt_abhaengig_von
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	[schritt1]		schrittnummer1
	schritt		schrittnummer2

Normale Abhängigkeiten zwischen Schritten (Abhängigkeiten innerhalb desselben Subworkflows) sind hierdurch festzulegen. Ein Abhängigkeits-Element beschreibt dabei, dass der Schritt mit Nummer "schritt1" von dem Schritt mit Nummer "schritt" abhängt, d.h. dass "schritt1" erst bearbeitet bzw. erledigt (abh. davon wie strikt die Abhängigkeiten für diesen Schritt gelten) werden kann, wenn "schritt" erledigt. Um sich die Angabe der Schrittnummer "schritt1" sparen zu können, muss das Abhängigkeits-Element ein Kind des schritt-Elements vom Schritt mit dieser Nummer sein.

Sollen die Schrittnummern der einzelnen Schritte automatische generiert werden, so müssen Aliase in den jeweiligen schritt-Elementen dafür eingeführt werden. definiert werden, die dann im Wert der hier vorkommenden Attribute "schritt1" und "schritt" mit "@(<aliasname>)" referenziert werden können (siehe schritt-Element oben).

workflow_schleifenstart-Elemente (i.d.R. als Kinder eines Schritt-Elementes für einen Schleifenbildungs-Schritt):

XML-Element:	workflow_schleifenstart	DB-Relation:	workflow_schleifen_start
XML-Attribute:	[workflow]	DB-Attribute:	workflownummer
	[schritt1]		schleifenschritt
	schritt		schrittnummer2

Ähnlich wie normale Abhängigkeiten zwischen Schritten, werden Schleifenstarts definiert.

Dabei beschreibt ein workflow_schleifenstart-Element, dass vom Schritt mit Nummer "schritt1" (sollte vom Typ "loop" sein) eine Schleife zum Schleifenstart-Schritt "mit Nummer "schritt" möglich ist, d.h.sind alle Abhängigkeiten für "schritt1" erfüllt, so kann statt diesen als "erledigt" zu markieren die Schleife neu geöffnet werden, d.h. der offen/erledigt-Status von "schritt" und allen (gemäß normaler Abhängigkeiten) (rekursiv) folgenden Schritten (bis zu "schritt1") wird wieder auf "offen" gesetzt". Hierbei sind pro Schleifenbildungs-Schritt mehrere Schleifenstart-Schritte möglich (nötig, falls die Schleifenstart-Schritt nicht untereinander Abhängigkeiten haben).

Die Schrittnummer "schritt1" muss nicht spezifiziert werden, falls das workflow_schleifenstart-Element Kind des schritt-Elements des Schrittes mit dieser Nummer ist. Wie bei abhaengigkeits-Elementen werden i.a. hier bei Verwendung automatischer Generierung von Schrittnummern Aliase für die Schrittnummern verwendet.

Der genaue Ablauf der Konvertierung läuft wie folgt:

1. Das XML-Struktur des XML-Eingabe-File wird geparst und ein Baum aus Elementen und Attributen im Speicher aufgebaut, wobei im Normalfall einem XML-Element ein Element im berechneten Baum entspricht. Die Attribute für ein solches Element werden dabei aus den XML-Attributen des ursprünglichen XML-Elements gewonnen. Konkret bedeutet dies:

   (a) XML-Attribute, derren Namen die Struktur “alias:<aliasname>” haben, werden nicht als Attribute im entstehenden Element, sondern als Alias-Definitionen für das im XML-Attributwert genannte Attribut diese Elements aufgefasst.

   (b) XML-Attribute, derren Namen die Struktur “aliased:<attributname>” haben, werden auch nicht unmittelbar mit diesem Attributamen in das Baum-Element übernommen, sonder mit dem Attributnamen “<attributname>”, wobei gespeichert wird, dass im Wert dieses Attributs Aliase für Attribute anderer Elemente vorkommen können, die später aufgelöst werden.

   (c) Alle sonstigen XML-Attribute werden einfach als Attribute des Baum-Elementes übernommen. Konflikte zwischen Baum-Attributen, die direkt mit ihrem Namen “<attributname>” als XML-Attribute vorkommen und als “aliased:<attributname>”, sollten vermieden werden.

   (d) Die XML-Element-Hierarchie wird im wesentlichen übernommen. Fast alle XML-Elemente werden zu Baum-Elementen. Es gibt nur eine Ausnahme:
   Um dem Benutzer das Editieren (mit nicht-XML-fähigen Editoren) von Attributwerten, die Sonderzeichen wie v.a. “<” und “>” enthalten (manche Module z.B. können SQL-Queries als Parameter haben und < und > dienen dort dem Verglich) zu erleichtern besteht aber die Möglichkeit einen Wert, der später im eingelesenen Element/Attribut-Baum ein Attribut eines Elementes darstellen soll, nicht als XML-Attribut des entsprechenden XML-Elementes, sondern als ein spezielles Kind-Element dieses XML-Elementes darzustellen. Der Element-Inhalt eines solchen speziellen XML-Elementes sollte einfach der gewünschte spätere Attributwert (als Text, d.h. kann v.a. mit “<![CDATA[“ und “]]>” umklammert sein, um beliebigen Text für das Attribut darin zu ermöglichen) sein um keine weiteren XML-Elemente enthalten. Der spezielle XML-Element-Name ist “A” und mit dem XML-Attribut “name” wird der Name des gewünschten Attributs angegeben:
   z.B.: '<schritt attributname1="wert1"> <A name="attributname2"> wert2</A> </schritt>' wäre gleichwertig zu '<schritt attributname1="wert1" attributname2="wert2"></schritt>'
   Allerdings müssen bei zweiten Lösung in wert2 sämtliche Zeichen, die nicht in XML-Attributwertenerlaubt sind (v.a. “<”,”>”) als XML-Charakterreferenzen geschrieben werden. Bei der ersten Lösung ist statt dessen in wert2 eine CDATA-Umklammerung des Textes, der die nicht erlaubten Zeichen enthält, möglich.
   Dem Benutzer bleibt überlassen, ob er diese spezielle Möglichkeit der Angabe von Baum-Attributen benutzt.

2. Ist der Element/Attribut-Baum aus der XML-Eingabe aufgebaut worden, werden bei Elementen fehlende Attribute, die aus der Struktur ersichtlich sind, wie oben beschrieben ergänzt.

3. Der um fehlende Attribute ergänzte Element/Attribut-Baum wird in SQL-INSERT-Statements umgewandelt. Dabei wird i.a. ein Element zu einem SQL-INSERT in eine DB-Relation und seine Attributwerte zu den Tupel-Attributwerten der Relation. (siehe Tabellen oben).

Die oben beschriebenen Regeln für die automatische Ergänzung von Attributen eines Elements, z.b. hierarchischer Strukturinformation oder Nummerierungen, sind in einer perl-Library, dem Skript “xml2sqlinsert-definitions-workflow.pl” im Ordner “Skripte” in der Variablen “@node_defaultvalues” definiert.

In dieser Library sind ebenso die Regeln für die Übersetzung der XML-Elemente in DB-Relationen bzw. XML-Attribute in DB-Attribute” in der Variablen “@node2table” zu finden.

Das Konverter-Skript “wfcreate.pl” übernimmt diesen ganzen Konvertierungsvorgang. Es liegt im Unterverzeichnis “Skripte”. Zur Lauffähigkeit des Skriptes muss das Perl-Paket “expat” (Version >= 1.95-1) zum Parsen von XML-Daten installiert sein. “wfcreate.pl “ wird mit der XML-Eingabe-Datei als Parameter aufgerufen und gibt auf die Standardausgabe die SQL-Anweisungen zum Abspeichern des neuen Workflows in der Datenbank aus. Diese Anweisungen können direkt verwendet werden, um den neuen Workflow anzulegen. Ein Beispiel für eine Workflow-Konfiguration befindet sich in der Datei “SQL/workflow_tgi.xml”.

4 Modulprogrammierung

4.1 Das Modulkonzept

Für den Aufbau der HTML-Seite eines Workflow-Schritts ist das PHP-Skript 'workflow.php' zuständig. Es erzeugt den Rahmen der HTML-Seite, der u. a. den Namen und den Status des Workflow-Schritts wiedergibt. Die Funktionalität des Workflow-Schritts ist in Modulen ausgelagert, die von 'workflow.php' innerhalb des Rahmens eingebunden werden. Dabei kann ein Workflow-Schritt ein oder mehrere Module benutzen. Die Namen der Module müssen bei der Workflow-Definition angegeben werden und finden sich in der Datenbank in der Relation 'workflow_schritt_modul' wieder. Jedem Modul entspricht ein PHP-Skript mit dem Namen 'module-<Modulname>.inc.php', das seinerseits eine Funktion 'mdoule_<Modulname>_main($control_args, $module_args)' enthalten muss. Die Arbeitsweise von 'workflow.php' und dessen Zusammenspiel mit den Modulen wurde bereits in den Abschnitten 3.4 und 3.5 ausführlich beschrieben.

Aus den Anforderungen an das Workflow-System (siehe Abschnitt 1.1) und der Analyse einiger Workflows konnten drei Arten von Funktionalitäten bestimmt werden, die von den Modul-Skripten erbracht werden müssen:

1. Die häufigste Aufgabe ist das Darstellen und Manipulieren von DB-Inhalten. Die Library 'lib-dbtable.inc.php' stellt Funktionen bereit, mit deren Hilfe solche Aufgaben in den Module realisiert werden können. Die Library wird in Abschnitt 4.3 näher beschrieben.
2. Um weniger privilegierten Personen den Zugriff auf die Datenbank zu ermöglichen, müssen temporäre Tabellen und Views eingerichtet und freigeschaltet werden. Hierbei bietet die Bibliothek 'lib-extmodul-control.inc.php' (siehe Abschnitt 4.4) Unterstützung.
3. Für bestimmte Tätigkeiten existieren bereits andere Skripte oder Programme, die aus der Workflow-Anwendung heraus aufgerufen werden sollen. Für solche Aufgaben kann in den Modulen auf die Library 'lib-file.inc.php' zurückgegriffen werden. Die Verwendung wird in Abschnitt 4.5 erklärt.

Ein Modul kann dabei mehrere, auch verschiedene Funktionalitäten enthalten. Die einzelnen Funktionsblöcke innerhalb eines Moduls werden im Folgenden Submodule genannt.

Wie generisch die einzelnen Module gehalten werden, bleibt dem Modulprogrammierer überlassen. In manchen Fällen kann es sinnvoll sein, ein sehr spezialisiertes Modul anzulegen. Dann sind alle für das Modul notwendigen Parameter fest im Modul-Skript eingebaut. Im anderen Extremfall kommen alle Parameter von außen. Für das Übergeben der Parameter an das Modul sind zwei Möglichkeiten vorgesehen:

1. Es können bereits bei der Workflow-Definition Parameter angegeben werden. Nach dem Parsen des XML-Files, das die Definition enthält, finden sich die Parameter in der Relation 'workflow_schritt_modul_param' der Datenbank wieder. Dort können sie nachträglich geändert und weitere eingefügt werden. Diese Parameter werden dem Modul-Skript beim Aufruf der Main-Funktion des Moduls im 2. Argument ($modul_param) übergeben. Hierbei handelt es sich um ein Array, das mit den Namen der Parameter indiziert ist.
2. Es können extra PHP-Skripte angelegt werden, die als Container für die Parameter dienen. Für diese Skripte gibt es folgende Namenskonvention: moduleconfig-<Name>[-<WorkflowNummer>].inc.php (<WorkflowNummer> ist optional). Im Modul-Skript kann der Name des Config-File mit der Funktion 'moduleconfig_filename($<Name>)' bestimmt werden. Die Funktion bietet den Vorteil, dass im Modul-Skript nur der Bestandteil <Name> des Config-File bekannt sein muss. Zunächst überprüft 'moduleconfig_filename', ob für die aktuelle Workflow-Nummer ein File mit dem Namen 'moduleconfig-<Name>-<WorkflowNummer>.inc.php' vorhanden ist. Falls nicht wird nach einem File mit dem Namen 'moduleconfig-<Name>.inc.php gesucht. Existiert auch diese Datei nicht, gibt die Funktion 'false' zurück. Nachdem der Name des gültigen Config-Files bestimmt ist, kann die Datei mit Hilfe der include-Funktion eingebunden werden.

```
if($include_name = moduleconfig_filename("KlausurAnmeldung"))
  include($include_name);
```

An dieser Stelle sei noch auf die Datei 'workflow-defines.inc.php' hingewiesen. Sie enthält einige für den Workflow zentrale Datei- bzw. Verzeichnisnamen, u. a. den Ort, an dem die Config-Files abgelegt sind.

- IMAGE_PREFIX: Image-Verzeichnis, in dem unter anderem nach den Gifs für die Buttons der DBTable-Tabellen gesucht wird.
- INCLUDE_PREFIX: Library- und Modul-Verzeichnis.
- STYLESHEET_LINK: CSS-Datei, die für die Workflow-Anwendung gelten soll.
- MODULECONFIG_INCLUDEPREFIX: Config-File-Verzeichnis.

4.2 Datenbankschema

4.2.1 Identifizierte Entities

Die im Fortgeschrittenen-Praktikum eingerichtete Datenbank umfasst zwei Teile. Zum einen ist dies der Bereich, der die notwendigen Steuerinformationen für die Workflow-Anwendung enthält. Diese ergeben sich aus der Definition eines Workflows und bestimmen beispielsweise welche Workflowschritte es gibt und welche Abhängigkeiten zwischen ihnen bestehen. Das zugehörige Datenbankschema wird in Abschnitt 3.2 beschrieben. Der andere Teil dient der Speicherung der Daten, die im Zusammenhang mit einer Lehrveranstaltung, einem FoPra oder einer Diplomarbeit anfallen. Diesen Daten sollen von der Anwendung dargestellt oder mit Hilfe der Anwendung manipuliert werden. Für die Erstellung des Datenbankschemas wurden folgende Entities identifiziert, die sich grob in drei Gruppen einteilen lassen.

- Lehrveranstaltungen: Vorlesung, Seminar, Praktikum, FoPra, Diplomarbeit und Oberseminar.
- Auf bestimmte Veranstaltungen bezogene Objekte: Übungsgruppe, Übungsblatt, Klausur, Aufgabe, Praktikumsgruppe, Seminarvortrag, Schein.
- Basisobjekte: Mitarbeiter, Mailadresse_Mitarbeiter, Student, Mailadresse_Student, Raum.

Manche der aufgeführten Entities sind dafür gedacht, Daten längerfristig zu History-Zwecken zu speichern. Dies sind FoPra, Diplomarbeit und Schein. Die Datenbank umfasst außerdem Entities, die zu einer Terminverwaltungsanwendung gehören. Diese war Bestandteil eines parallelen FoPra. Einige der genannten Entities (Mitarbeiter, Student, Oberseminar etc.) werden von beiden Anwendungen benutzt. Der überschneidende Teil des DB-Schemas wurde von den beiden FoPra-Gruppen gemeinsam entwickelt.

4.2.2 Oberklassen und relationale Datenbanken

Aus der großen Zahl der Entities ergibt sich eine noch größere Anzahl an Beziehungen. Dies wären beispielsweise "Student besucht Vorlesung", "Student besucht Übung", "Student macht Praktikum", "Student macht FoPra" oder "Student macht Diplomarbeit". Um die Zahl der Beziehungen zu verringern, wurde zeitweise daran gedacht, durch Generalisierung eine Oberklasse für die verschiedenen Veranstaltungsarten zu bilden. Dadurch hätte man gleichartige Informationen in einer Relation zusammenfassen können. Mögliche Attribute für diese Oberklasse hätten die verschiedenen offiziellen Verzeichnisnummern der Veranstaltung, eine gemeinsame laufende Nummer, Name der Veranstaltung, Bemerkung, Semester etc. sein können. Bei vielen Beziehungen hätte dann der Bezug auf die Oberklasse ausgereicht. Ein weitere Oberklasse hätte aus den Mitarbeiteren des Lehrstuhls und Tutoren gebildet werden können.

Im Gegensatz zu objektorientierten Datenbanken kennen relationale Datenbanken jedoch keine Vererbung oder Generalisierung. Die im Entity-Relationship-Diagramm mögliche "IS-A"-Beziehung lässt sich in einer relationalen Datenbank nicht ordentlich modellieren. In den Relationen, die Spezialisierungen repräsentieren würden, ist nicht die volle Information verfügbar. "Mit anderen Worten, die Vererbung ist nicht realisiert" [Kemper/Eickler S. 70]. Die vollständige Information würde man erst durch einen Join mit der Oberklasse erhalten. Generalisierungen werden in relationalen Datenbanken hingegen oft durch Views bewerkstelligt. Views sind im 3-Schichten-Modell allerdings Bestandteile der externen und nicht der internen (logischen) Schicht.

4.2.3 Umsetzung der ER-Diagramme in der relationalen Datenbank

Im Anhang finden sich einige ER-Diagramme, die das Datenbankschema wiedergeben. Einige Stellen des Datenbankschemas bedürfen jedoch noch einer zusätzlichen Erklärung. Die Beziehungen "Mitarbeiter beaufsichtigt Klausur" und "Student beaufsichtigt Klausur" wurden durch eine gemeinsame Relation 'personal_beaufsichtigt_klausur' realisiert. Diese Relation enthält als Schlüssel 'vorlesungnummer', 'klausurnummer' (beide Primärschlüssel der Relation 'vorlesung'), 'aufsicht_istmitarbeiter' (entscheidet, ob es sich um einen Mitarbeiter oder Studenten handelt) und 'aufsicht_nummer' (Primärschlüssel aus 'mitarbeiter' bzw. 'student'). Dabei handelt es sich nicht um eine Generalisierung, da keine Oberklasse gebildet wird, sondern nur zwei gleichartige Beziehungen zusammengefasst werden. Ähnliches gilt für die Beziehungen "Übungsblatt hat Aufgabe" und "Klausur hat Aufgabe". Hier existiert jedoch keine eingene Beziehungsrelation. Da es sich jeweils um 1:n-Beziehungen handelt, wurde die Beziehung durch Fremdschlüsselimport in die Relation 'aufgabe' realisiert.

Die in den ER-Diagrammen aufgeführten Entities "Vorlesungspersonal", "Vorlesungsteilnehmer", "Seminarteilnehmer", "Praktikumspersonal" und "Praktikumsteilnehmer" wurden nicht durch eigene Relationen realisiert. Im ER-Diagramm sind sie jedoch notwendig, um einige Beziehungen (zum Beispiel "Vorlesungspersonal betreut Übungsgruppe") ordentlich zu modellieren. In der Datenbank wird das Entity "Vorlesungspersonal" durch die Relation "arbeitet_fuer_vorlesung" abgedeckt. Dies ist eigentlich eine Beziehungsrelation zwischen 'student' und 'vorlesung' bzw. 'mitarbeiter' und 'vorlesung'. (Hier gilt das, was im letzten Absatz dargestellt wurde.) Bei dem Entity "Praktikumspersonal" verhält es sich genau so. Die Entities "Vorlesungsteilnehmer" und "Praktikumsteilnehmer" werden durch die Relationen 'student_besucht_vorlesung' bzw. 'student_macht_praktikum' abgedeckt. Auf diesen Beziehungsrelationen werden dann selbst wieder Beziehungen aufgesetzt. Dies ist beispielsweise bei 'student_besucht_uebung' der Fall. Wie im ER-Diagramm vorgegeben, wird dadurch garantiert, dass ein Übungsteilnehmer auch Vorlesungsteilnehmer sein muss. Das Entity "Seminarteilnehmer" fällt mit der Beziehung "Seminarteilnehmer hält Seminarvortrag" zusammen und findet sich in der Relation 'student_haelt_seminarvortrag' wieder. Auf diese Relation bezieht sich außerdem die Relation 'student_anwesend_bei_vortrag'.

Die im ER-Diagramm dargestellte Beziehung "Vorlesungsteilnehmer besucht Übung" kommt eigentlich mehrmals vor. Ein Student kann mehrere Präferenzen für die Übungsteilnahme angeben, bevor er einer Übungsgruppe endgültig zugeteilt wird. Schaut man sich die Relation 'student_besucht_uebung' an, stellt man fest dass hier neben der genannten Beziehung auch mehrere "Präferenz"-Beziehungen festgelegt sind.

Die Beziehungen "Student nimmt teil an Praktikum" und "Praktikumsteilnehmer gehört zu Praktikumsgruppe" scheinen redundant zu sein. Da jedoch die Anmeldung zu einem Praktikum und die Gruppeneinteilung oft zeitlich weit auseinander liegen, ist diese Modellierung sinnvoll. Außerdem sind auch Praktikas denkbar, bei denen keine Gruppen gebildet werden.

Das Entity-Relationship-Diagramm für das FoPra gilt in gleicherweise für Diplomarbeiten. Das Relationship "Student macht Diplomarbeit" ist als eigene Relationen ausgeführt, auch wenn es sich hierbei aller Voraussicht nur um eine 1:n-Beziehung handelt. Da die Relation für das Entity "FoPra" Daten längerfristig speichern soll (History), die Relation 'student' aber aus datenschutzrechtlichen Gründen regelmäßig bereinigt werden muss, ist eine eigene Beziehungsrelation an dieser Stelle sinnvoll. Die Diplomarbeit kann dann auch unabhängig vom Student existieren. In der Beziehungsrelation kann trotzdem das Attribut 'studentnummer' mit der Einschränkung

"NOT NULL" versehen werden. Die anderen Beziehungen müssen sowieso als eigene Relationen ausgeführt werden, da von m:n-Beziehungen ausgegangen werden muss.

4.3 DB-Table-Submodule

4.3.1 Einleitung

In einem Workflow-Schritt können Inhalte einer Datenbank dargestellt und darüber hinaus Datenmanipulationen vorgenommen werden. Dazu sollen in den Modul-Skripten die Funktionen 'modul_tabelle' und 'modul_tabelle_list' verwendet werden, die von der Library 'lib_dbtable.inc.php' zur Verfügung gestellt werden. Die Bibliothek 'lib_dbtable' greift ihrerseits auf verschiedene andere Bibliotheken zurück, wobei zwei hervorzuheben sind:

- 'lib_sql.inc.php': Diese Library stellt DBMS-unabhängige Funktionen für den DB-Zugriff bereit. Beispielhaft sei die Funktion 'db_query ($query)' erwähnt. Beim Aufruf führt diese Funktion die übergebene SQL-Anfrage durch und liefert einen Result-Handler zurück. Mit Hilfe des Result-Handlers kann dann auf die Ergebnisse der Anfrage zugegriffen werden.
- 'lib_html.inc.php': Bei der HTML-Ausgabe ist immer wieder eine Folge von gleichen oder ähnlichen HTML-Tags notwendig. Die lib_html-Library bietet Funktionen, um bestimmte Tag-Folgen zu generieren. Dabei können diesen Funktionen Daten übergeben werden, die von den Tags umschlossen werden sollen. Die Library bietet dabei Funktionen für einfache, aber auch komplexe Tag-Folgen. Beispielsweise kann mit der Funktion 'html_table ($data, ...)' eine komplette HTML-Tabelle erzeugt werden, die die im Array $data übergebenen Inhalte darstellt.

Die Library 'lib_dbtable.inc.php' stellt dabei das Verbindungsglied dar. Sie extrahiert Daten aus der Datenbank, stellt diese Daten sinnvoll dar, bietet Interaktionsmöglichkeiten zur Datenmanipulation und führt diese Datenmanipulationen schließlich durch. Die Aufgaben im einzelnen:

- Die Library führt DB-Anfragen durch, die von den Modul-Skripten übergeben werden.
- Dabei werden die Anfragen zuvor einer einfachen Überprüfung unterzogen und falls nötig ergänzt.
- Im Falle einer SELECT-Anfrage wird das Ergebnisse in eine HTML-Tabelle eingebaut und ausgegeben.
- Da die Ergebnismenge einer SELECT-Anfrage sehr groß sein kann, bietet 'lib_dbtable.inc.php' folgende Unterstützungen:
  - Die Daten können seitenweise ausgegeben werden. In die HTML-Ausgabe werden Links integriert, um in den Seiten zu blättern.
  - Man hat die Möglichkeit die Ergebnismenge aus der Workflow-Anwendung heraus einzuschränken.
  - Innerhalb der Ergebnismenge kann aus der Workflow-Anwendung heraus nach verschiedenen Kriterien gesucht werden.
  - Der Anwender kann die Sortierreihenfolge der Ergebnisse beeinflussen.
- Für die Datenmanipulation werden Links oder Buttons in die HTML-Ausgabe eingefügt, mit deren Hilfe Aktionen ausgelöst werden können, um ein oder mehrere Datentupel hinzuzufügen, zu ändern oder zu löschen.

Für die Darstellung und Datenmanipulation verwendet die Library Default-Werte, die in den Modul-Skripten jedoch angepasst werden können. Die Darstellungsparameter, die Parameter für DB-Queries und die Parameter für die zu unterstützenden Aktionen werden alle in einem großen

Array $modul_param zusammengefasst. Dieses Array wird der Funktion 'modul_tabelle ($modul_param)' übergeben.

In einem Modul-Skript können durchaus mehrere DB-Table-Submodule enthalten sein. Dabei sind aber eventuelle Datenabhängigkeiten zwischen den Submodulen zu berücksichtigen. Zum Verständnis stelle man sich folgendes Beispiel-Modul vor: Zuerst werden in einer Tabelle, die einer bestimmten Veranstaltung zugeordneten Mitarbeiter dargestellt. Darunter werden alle verfügbaren - auch die nicht zugeteilten - Mitarbeiter des Lehrstuhls ausgegeben, um gegebenenfalls einen weiteren Mitarbeiter der Veranstaltung zuweisen zu können. Die beiden Mitarbeitertabellen werden durch zwei getrennte Aufrufe der Funktion 'modul_tabelle ($modul_param)' erzeugt. Angenommen die Tabelle mit allen Mitarbeitern bietet die Möglichkeit die Namen der Mitarbeiter zu ändern, um z. B. einen Schreibfehler auszubessern. Der Mitarbeiter Maier, der auch der Veranstaltung zugeteilt ist - also in beiden Tabellen auftaucht - schreibt sich korrekt Mair. Die Datenänderung wird in der unteren Tabelle vorgenommen. Nach der Bestätigung wird die HTML-Seite mit dem Modul und den beiden Tabellen neu aufgebaut. Die Datenänderung geschieht erst beim Aufbau der zweiten Tabelle, also mit dem zweiten Aufruf von 'modul_tabelle ($modul_param)'. Die erste Tabelle weiß deshalb von der Datenänderung noch nichts, und der Name erscheint dort nach wie vor in der falschen Schreibweise. Um dieses Problem zu umgehen bietet die Funktion ein zweites Argument: 'modul_tabelle ($modul_param, $update_only)'. Wird in $update_only 'true' übergeben (Standardwert: 'false'), werden nur die Datenmanipulationen auf der Datenbank durchgeführt, es erfolgt aber keine HTML-Ausgabe. Um das Problem zu umgehen, würde man die Funktion 'modul_tabelle ($modul_param, $update_only)' dreimal aufrufen:

1. Aufruf für die zweite Tabelle ohne HTML-Ausgabe ($update_only=true).
2. Aufruf für die erste Tabelle mit HTML-Ausgabe ($update_only=false).
3. Erneuter Aufruf für die zweite Tabelle diesmal mit HTML-Ausgabe ($update_only=false).

Zur Vereinfachung kann auch die Funktion 'modul_tabelle_list ($modul_param1, $modul_param2, ...)' verwendet werden. Dieser Funktion können in einer variablen Argumentliste die Parameter-Arrays für jedes DB-Table-Submodul übergeben werden. Die Funktion ruft ihrerseits für jedes Submodul modul_param zweimal auf - einmal zum Update der Datenbank und einmal für die HTML-Ausgabe.

4.3.2 Zusammensetzung des Parameter-Array $modul_param

Den Funktionen 'modul_tabelle' bzw. 'modul_tabelle_list" werden alle notwendigen Informationen im Array $modul_param übergeben. Die Bestandteile des Array sind in Tabelle 1 aufgeführt. Die meisten Bestandteile sind selbst wieder Arrays:

- required_attribs: Dieses Array ist nach den Attributnamen indiziert. Es bietet sich beispielsweise an die Vorlesungsnummer auf diese Weise fest vorzugeben. Für eine Vorlesung mit der Nummer 10 könnte man folgende Vorgabe machen: required_attribs = array (vorlesungnummer => 10). Somit ist sichergestellt, dass sich alle DB-Anfragen auf die richtige Vorlesung beziehen. In Zusammenhang mit dem Eintrag 'anzeige [appearance] [nur_bei_link_aktion]]' (siehe 4.3.9) kann als Wert für ein Attribut auch folgendes Konstrukt verwendet werden: required_attribs = array (eigenesfeld => 'LINK:tabellenid.fremdfeld'). Auf diese Weise hat man die Möglichkeit auf den Wert (des Feldes <fremdfeld>) eines anderen DB-Table-Submoduls (durch tabellenid referenziert) zuzugreifen. Dies ist aber nur möglich, wenn der Benutzer in jenem Submodul die Aktion 'aendern' oder 'loeschen' (siehe 4.3.6) ausgelöst hat.
- anzeige, aendern, loeschen, neues_tupel: Die Parameter von anzeige beziehen sich auf die Haupttabelle. Änderungen und Löschungen können direkt in der Haupttabelle durchgeführt werden. Es ist jedoch auch möglich für diese Aktionen zunächst eine zusätzliche Tabelle aufzubauen. In dieser Zwischentabelle können weitere Attribute angezeigt werden, die in der Haupttabelle aus Gründen der Übersichtlichkeit ausgeblendet wurden. Beim Ändern eines

Schlüssel	Beschreibung
tabellenid	ID des Submoduls (mandatory)
leittabelle	Name der DB-Relation, auf die zugegriffen werden soll (mandatory)
required_attribs	Array mit Attribut-Werte-Paaren, nach denen bei allen Anfragen selektiert werden soll.
anzeige	Array mit Parametern für die Haupttabelle
aendern	Array mit Parametern für die Ändern-Tabelle
loeschen	Array mit Parametern für die Löschen-Tabelle
neues_tupel	Array mit Parametern für das Formular zum Erstellen eines neuen Tupels
reihenfolge	Array, in dem die Ausgabereihenfolge der Haupttabelle, der Ändern-Tabelle, der Löschen-Tabelle und dem Formular für das neue Tupel verändert werden kann.

Tabelle 1: Bestandteile von $modul_param

Datentupels könnten in der Zwischentabelle zusätzliche Attribute zum Ändern angeboten werden. Beim Löschen kann die Zwischentabelle als Sicherheitsabfrage verstanden werden. Für das Erzeugen eines neuen Tupels wird ein Formular aufgebaut, in das die neuen Daten eingegeben werden können. Die vier Arrays benutzen im Wesentlichen alle die selben Parameter.

- reihenfolge: Standardmäßig werden die Zwischentabellen für das Ändern und Löschen bzw. das Formular zum Erzeugen eines neuen Tupels vor der Haupttabelle ausgegeben. Mit Hilfe dieses Arrays kann die Reihenfolge beeinflusst werden. Beispielsweise würde $reihenfolge = array (neues_tupel, anzeige, aendern, loeschen) bewirken, dass nur das Formular für ein neues Tupel vor der Haupttabelle (anzeige) ausgegeben wird. Die Zwischentabellen zum Ändern und Löschen werden hingegen nach der Haupttabelle ausgegeben.

Schon alleine mit den Einträgen für tabellenid und leittabelle lässt sich ein Aufruf der Funktion 'modul_tabelle' durchführen. Da sich ein Workflow in der Regel auf nur eine Veranstaltung bezieht, ist es empfehlenswert mit dem Eintrag required_attribs eine entsprechende Einschränkung vorzunehmen. Das folgende Beispiel erzeugt eine Tabelle mit allen Übungsgruppen zur Vorlesung mit der Vorlesungsnummer 10:

```
$modul_param = array (
  tabellenid => ''uebungsgruppe'',
  leittabelle => ''uebungsgruppe'',
  required_attribs => array(
    vorlesungnummer => 10
  )
);
modul_tabelle_list($modul_param);
```

4.3.3 Die Einträge der Arrays $anzeige, $neues_tupel, $aendern und $loeschen im Überblick

Die Arrays $anzeige, $neues_tupel, $aendern und $loeschen sind Bestandteile des Arrays $modul_param, das den Funktionen 'modul_tabelle' bzw. 'modul_tabelle_list' als Argument übergeben wird. Mit den Parametern dieser vier Arrays werden Teilbereiche eines DB-Table-Submoduls gesteuert:

- $anzeige: Enthält Parameter für die Haupttabelle. In der Haupttabelle werden die wesentlichen Daten eines DB-Table-Submoduls dargestellt. Es ist möglich in der Haupttabelle Änderungen und Löschungen vorzunehmen. Außerdem können Datentupel der Haupttabelle ausgewählt werden, um sie in andere Relationen einzufügen.

- $neues_tupel: Enthält Parameter für das Formular, über das neue Daten eingegeben werden können. Das Formular wird nicht immer angezeigt. Die Anzeige kann beispielsweise ausgelöst werden, indem in der Haupttabelle der Button neues_tupel angeklickt wird.
- $aendern: Enthält Parameter für die Ändern-Tabelle. Diese Tabelle wird nicht immer angezeigt. Die Anzeige kann beispielsweise ausgelöst werden, indem in der Haupttabelle auf einen Link zur Änderung eines bestimmten Tupels geklickt wird. Alternativ können Änderungen auch direkt in der Haupttabelle vorgenommen werden. Unter Umständen ist es jedoch sinnvoll die Zwischen-Tabelle zu verwenden, wenn beispielsweise aus Gründen der Übersichtlichkeit in der Haupttabelle nicht alle Attribute angezeigt werden. Weiterhin kann es möglich sein, dass in der Haupttabelle die Datentupel gruppiert angezeigt werden. In der Zwischentabelle könnte man statt dessen die Einzeltupel einer Gruppe anschauen und verändern.
- $loeschen: Enthält Parameter für die Löschen-Tabelle. Diese Tabelle wird nicht immer angezeigt. Die Anzeige kann beispielsweise ausgelöst werden, indem in der Haupttabelle auf einen Link zum Löschen eines bestimmten Tupels geklickt wird. Alternativ können Löschungen auch direkt in der Haupttabelle vorgenommen werden. Es könnte jedoch sinnvoll sein die Zwischentabelle als Sicherheitsabfrage und Überprüfung zu verwenden. In der Zwischentabelle können beispielsweise mehr Attribute eines Datentupels angezeigt werden als in der Haupttabelle. Außerdem könnten in der Haupttabelle die Daten gruppiert sein. Dann ist es sicher sinnvoll, vor dem Löschen die Tupel noch einmal einzeln zu betrachten.

Die vier Arrays enthalten Einträge, mit deren Hilfe die Darstellung der Daten, die Datenmanipulation und die Ablaufsteuerung beeinflusst werden kann. Die Eintäge sind in Tabelle 4.3.3 zusammengefasst. Die Arrays benutzen im Wesentlichen die selben Einträge, einzelne Einträge machen jedoch nicht bei allen Arrays Sinn. Tabelle 3 zeigt welche Einträge in welchen Arrays zur Verfügung stehen.

Eintrag	Beschreibung
query	In den Tabellen werden standardmäßig alle Attribute und alle Datentupel (evtl. Selektion durch $required_attribs) aus der Leittabelle dargestellt. Mit dem Array $query kann die SELECT-Anfrage verfeinert werden. Beispielsweise können zur Leittabelle Attribute aus einer anderen Tabelle durch einen Join ergänzt werden.
cols	Manchmal möchte man nicht alle Attribute aus dem Ergebnis der SELECT-Anfrage darstellen. Im Array $cols kann deshalb eine Liste der Attribute angegeben werden, die in der HTML-Ausgabe erscheinen sollen. Die Attribute erscheinen bei der Ausgabe in der Reihenfolge in der sie im Array $cols stehen.
aktionen	Array, in dem die Aktionen (Ändern, Löschen, ...) angegeben werden, die im Submodul möglich sein sollen.
table_param	Array, mit dem die Tabellen-Darstellung beeinflusst werden kann. Es können beispielsweise die verwendeten Farben angepasst werden.
form_param	Array, mit dem die Formular-Darstellung beeinflusst werden kann. Es kann beispielsweise die Größe eines Formularfeldes beeinflusst werden.
appearance	Array, mit dem weitere Darstellungs-Eigenschaften beeinflusst werden können. Es kann beispielsweise angegeben werden, wieviel Zeilen auf einmal ausgegeben werden sollen.

Eintrag	Beschreibung
usertext	Array, das Texte aufnehmen kann, die nach oder vor der Tabelle bzw. dem Formular ausgegeben werden sollen. Darin können beispielsweise Kurzbeschreibungen zu den Tabellen bzw. Formularen übergeben werden.
prequeries	Array, das Queries aufnehmen kann, die zusätzlich vor einer Datenmanipulation ausgeführt werden.
composed_attribs	Array, mit dessen Hilfe zusammengesetzte Attribute definiert werden können. Dies ist bei bestimmten, komplexen Datenmanipulationen notwendig.
values	Array, in dem Attribut-Werte-Paare vorgegeben werden können, die beim Hinzufügen eines neuen Datentupels in eine Relation als Standardwerte verwendet werden.
wrapper	Häufig ist es notwendig, dass ein neues Datentupel nich nur in eine Relation, sondern in mehrere Relationen eingetragen wird. Mit diesem Array kann für jede Relation vorgegeben werden, welche Bestandteile des Datentupels in welche Relation eingefügt werden sollen.
wrap_param	Beim gleichzeitigen Hinzufügen von Daten in mehrere Relationen sind zusätzliche Steuerungsinformationen nötig. Was soll beispielsweise geschehen, wenn das Einfügen in eine Relation fehlgeschlagen hat? Dieses Verhalten kann mit diesem Array beeinflusst werden.
dataconversion	Oftmals ist die Repäsentation eines Wertes in der Datenbank für die Darstellung nicht geeignet. Insbesondere für Zeit- und Datumsformate ist eine angepasste Darstellung oft sinnvoll. Mit diesem Array kann die Darstellung entsprechend beeinflusst werden.
aendern_cols	Array, das eine Liste der Attribute enthält, die verändert werden dürfen.
ziellink	Kann eine URL enthalten, die statt der Standardseite beim Klick auf einen Link oder einen Button aufgerufen wird.
tabelle	Kann den Namen einer Relation enthalten, die statt der Leittabelle als Standardrelation verwendet werden soll.

Überblick über die möglichen Einträge in den Arrays $anzeige, $neues_tupel, $aendern und $loeschen

4.3.4 Der Eintrag query im Detail

Der Eintrag query steht in $anzeige, $aendern und $loeschen zur Verfügung. query ist selbst ein Array, das mehrere Anfragen enthalten kann. Jede Anfrage ist ein Eintrag in query. Die Anfragen werden vor der Ausführung des Query mit UNION ALL verknüpft. Die einzelnen Anfragen selbst sind wiederum als Arrays (mit Hashes) organisiert, wobei die einzelnen Bestandteile (Tabelle 4) den Sprachelementen von SQL entsprechen. Die Bestandteile werden vor der Ausführung des Query in einen SQL-String umgewandelt. Der Syntax entspricht dem von SQL.

Das Beispiel aus 4.3.2 könnte man so erweitern dass, zusätzlich zur Tutornummer auch der Name des Tutors angezeigt wird.

```
$modul_param = array (
  tabellenid => ``uebungsgruppe'',
  leittabelle => ``uebungsgruppe'',
  required_attribs => array(
    vorlesungnummer => 10
```

Eintrag	anzeige	neues_tupel	aendern	loeschen
query	ja	nein	ja	ja
cols	ja	ja	ja	ja
aktionen	ja	nein	nein	nein
table_param	ja	ja	ja	ja
form_param	ja	ja	ja	ja
appearance	ja	ja	ja	ja
usertext	ja	ja	ja	ja
prequeries	ja	ja	ja	ja
composed_attribs	ja	nein	nein	nein
values	ja	ja	nein	nein
wrapper	ja	ja	nein	ja
wrap_param	ja	ja	nein	nein
dataconversion	ja	ja	ja	ja
aendern_cols	ja	ja	ja	nein
ziellink	ja	ja	ja	ja
tabelle	nein	nein	ja	nein

Tabelle 3: Welche Einträge in welchen Arrays?

Bestandteil	Beschreibung	Standardwert
SELECT	String, der die für die Projektion vorgesehenen Attribute (durch Kommas getrennt) enthält.	Leittabelle definiert: leittabelle.* sonst: *
FROM	String, der die zu verwendenden Relationen (durch Kommas getrennt) enthält.	Leittabelle definiert: leittabelle sonst: Fehler
WHERE	String, der die Selektionen (durch Kommas getrennt) enthält	$requierd_attribs, falls definiert
GROUP	String, der die Attribute (durch Kommas getrennt) enthält, nach denen gruppiert werden soll.	
ORDER	String, der die Attribute (durch Kommas getrennt) enthält, nach denen sortiert werden soll.	
REST	String, der beim Umwandeln in den SQL-String am Ende angefügt wird. Hierin können weitere SQL-Sprachelemente aufgenommen werden, falls nötig.	
WHERE_EXT	Array, das Informationen für die Gestaltung der Benutzer-Suche enthält. Steht nur in $anzeige zur Verfügung.	

Tabelle 4: Bestandteile eines Query-Arrays

```
    ),
    anzeige => array (
      query => array (
        array (
          SELECT => "u.*, m.vorname, m.name",
          FROM => "uebungsgruppe u, mitarbeiter m",
          WHERE => "u.tutor_istmitarbeiter<>'0'
            AND u.tutor_nummer=m.mitarbeiternummer",
        ),
        array (
          SELECT => "u.*, s.vorname, s.name",
          FROM => "uebungsgruppe u, student s",
          WHERE => "u.tutor_istmitarbeiter='0'
            AND u.tutor_nummer=s.studentnummer",
        ),
      ),
    ),
  );
  modul_tabelle_list($modul_param);
```

Da sowohl ein Mitarbeiter, als auch ein Student als Tutor für eine Übungsgruppe arbeiten kann, werden im query-Array zwei Teilanfragen formuliert. In der ersten wird ein Join mit der Relation 'mitarbeiter' vorgenommen, in der zweiten mit der Relation 'student'.

Im Zusammenhang mit Gruppierungen ist folgendes zu beachten: Normalerweise werden die einzelnen Ergebnistupel des Query durch die Primärschlüssel der Leittabelle gekennzeichnet. Wird beispielsweise die Aktion Ändern ausgelöst, kann das zu ändernde Tupel von der Anwendung durch den gespeicherten Primärschlüssel identifiziert werden. Da bei einer Gruppierung unter Umständen auch der Primärschlüssel einbezogen wird, wird hier ein anderer Mechanismus zur Identifizierung der einzelnen Tupel verwendet. Die Anwendung merkt sich die einzelnen Ergebnistuppel an Hand der Attribute der Leittabelle, nach denen gruppiert wurde. Das folgende Beispiel soll das Ändern der Mailadressen eines Student ermöglichen. Da ein Student mehrere Mailadressen haben kann, wird in der Haupttabelle nach den Mailadressen gruppiert. Dabei kommt es auf das Attribut 'm.studentnummer' an. An Hand dieses Attributs weiß die Anwendung, welche Mailadressen geändert werden sollen.

```
  $modul_param = array (
    tabellenid => "mailadressen",
    leittabelle => "mailadressenstudent",
    anzeige => array (
      query => array (
        array (
          SELECT => "m.studentnummer, s.vorname, s.name",
          FROM => "mailadressenstudent m, student s",
          WHERE => "m.studentnummer=s.studentnummer",
          GROUP => "m.studentnummer, s.vorname, s.name",
        ),
      ),
      aktionen => array (aendern => 1),
    ),
  );
  modul_tabelle_list($modul_param);
```

Die Workflow-Anwendung bietet über ein Formular dem Benutzer die Möglichkeit die unter Umständen sehr umfangreiche Ergebnismenge besser zu beherrschen. Er kann über dieses Formular nach bestimmten Ergebnistupeln suchen oder die Ausgabe entsprechend einschränken. Dabei stehen als Operatoren <, <=, >=, >, =, <> und das aus SQL bekannte LIKE (mit den Wildcards '%' und '_') zur Verfügung. Darüberhinaus kann der Benutzer die Sortierreihenfolge beeinflussen, indem er die Reihenfolge der Sortierkriterien vorgibt und für die einzelnen Kriterien auf- oder absteigende Sortierung auswählt.

Der Modulprogrammierer kann über das Array WHERE_EXT die Such- und Sortiermöglichkeiten aktivieren und beeinflussen. Das Array WHERE_EXT verwendet folgende Schlüssel:

- Die Aliasnamen der Attribute nach denen gesucht, eingeschränkt und sortiert werden soll. Als Wert reicht in der Regel folgender String aus: cmp(tupelvar.attributname). Wichtig ist, dass als Schlüssel der Alias, im Wert der Attributname verwendet werden muss.
- Ein Präfix 'op_' konkateniert mit einem Aliasnamen von oben. Als Wert kann man dann einen Operator vorgeben, der als Standardoperator für das entsprechende Attribut im Suchformular erscheinen soll. Ohne einen solchen Eintrage im WHERE_EXT-Array wird '=' als Standardoperator verwendet.

Die Funktion cmp(tupelvar.attributname) ist gleichbedeutend mit dem String 'tupelvar.attributname $(op_aliasname) $(aliasname)'. $(op_aliasname) und $(aliasname) wird zur Laufzeit durch den vom Anwender ausgewählten Operator bzw. durch das vom Anwender eingegebenen Suchkriterium ersetzt. Der Modulprogrammierer kann statt cmp(...) einen entsprechenden String auch direkt eingeben. Denkbar wären folgende Varianten:

- 'tupelvar.attributname = $(aliasname)': So wird als Operator immer '=' verwendet.
- 'tupelvar.attributname $(op_aliasname) $(aliasname2)': So kann auf den Wert eines anderen Suchkriteriums Bezug genommen werden.

(tupelvar, attributname, aliasname etc. sind natürlich durch konkrete Werte zu ersetzen.)

Im folgenden ein kurzes Beispiel, das das Suchen in der Relation student nach Vor- und Nachname ermöglicht:

```
$modul_param = array (
  tabellenid => "student",
  leittabelle => "student",
  anzeige => array (
    query => array (
      array (
        SELECT => "vorname, name as nachname,
          matrikelnummer, hauptfach",
        WHERE_EXT => array (
          vorname => "cmp(vorname)",
          nachname => "cmp(name)",
        ),
      ),
    ),
  ),
);
modul_tabelle_list($modul_param);
```

Da für das Attribut 'name' der Alias 'nachname' im SELECT-Teil verwendet wurde, muss dies im WHERE_EXT-Teil entsprechend berücksichtigt werden.

4.3.5 Der Eintrag cols im Detail

Das Array $cols enthält eine Liste der Felder, die in der HTML-Ausgabe erscheinen sollen. Es ist zu unterscheiden, ob $cols in anzeige, aendern und loeschen oder in neues_tupel verwendet wird:

- anzeige, aendern, loeschen: $cols bezieht sich auf die Ergebnismenge der SQL-Anfrage, wie sie in $query (siehe 4.3.4) angeben wurde. In der Liste sind falls definiert die Aliasnamen, sonst die Attributnamen zu verwenden. Falls ein spezielles Datacell-Format in $table_param (siehe) definiert wurde, kann auch der entsprechende Array-Schlüssel in $cols eingefügt werden.
- neues_tupel: $cols bezieht sich auf $values (siehe Abschnitt 4.3.13); es erscheinen nur die Formularfelder, die in $cols angegeben sind.

Neben der Auswahl der Felder wird mit $cols auch die Reihenfolge der Felder bei der Ausgabe beeinflusst. Das folgende Beispiel gibt eine Tabelle mit den Namen der Studenten samt Anschrift aus:

```
$modul_param = array (
  tabellenid => "student",

 leittabelle => "student",
  anzeige => array (
    query => array (
      array (
        SELECT => "s.name as nachname, s.*",
        FROM => "student s",
      ),
    ),
    cols => array("vorname", "nachname", "anschrift"),
    table_param => array(
      datacell_format => array(
        anschrift => "$[strasse], $[postleitzahl] $[ort]",
      ),
    ),
  ),
);
modul_tabelle_list($modul_param);
```

Im Beispiel wird statt 'name' der Alias 'nachname' verwendet. 'anschrift' wird als besonderes Datacell-Format in 'table_param' definiert. Es fasst die Attribute 'strasse', 'postleitzahl' und 'ort' zusammen.

(Es ist zu beachten, dass alle Attributnamen klein geschrieben werden.)

4.3.6 Der Eintrag aktionen im Detail

Der Eintrag 'aktionen' steht nur in 'anzeige' zur Verfügung. 'aktionen' ist als Array aufgebaut und bestimmt, welche Aktionen im Submodul ausgelöst werden können. Tabelle 5 zeigt die möglichen Aktionen. Das Array enthält als Schlüssel die Namen der Aktionen und als Werte '1' oder '0'. Bei '1' ist die entsprechende Aktion verwendbar, bei '0' nicht.

Bemerkungen:

- inline_aendern: Die änderbaren Felder werden in 'aendern_cols' (siehe) an 'modul_tabelle' übergeben. Ist 'aendern_cols' nicht definiert sind alle Nicht-Schlüssel-Attribute der Leittabelle änderbar - Ausnahme: Felder bei denen ein Aliasname definiert wurde.
- hinzufuegen(_multi), verschieben(_multi): Die Relationen in denen die Einfügungen vorgenommen werden sollen, sind durch den Eintrag 'wrapper' (siehe) definiert.
- Beispiel für hinzufuegen(_multi): Ein Student soll an einer Vorlesung teilnehmen. Dazu wird in der Tabelle 'student' ausgewählt und in die Tabelle 'student_besucht_vorlesung' eingetragen.

Aktion	Darstellung	Auswirkung
loeschen	Zusätzliche Spalte 'Aktionen'. Spalte enthält in jeder Zeile verlinktes Image 'loeschen2.gif'	Klick auf das Image bewirkt zunächst nur das Einblenden der Löschen-Tabelle. Erst Klick auf den OK-Button der Löschen-Tabelle führt zum Löschen des Datentupels.
aendern	Zusätzliche Spalte 'Aktionen'. Spalte enthält in jeder Zeile verlinktes Image 'bearbeiten2.gif'.	Klick auf das Image bewirkt das Einblenden der Ändern-Tabelle. Nach einen Klick auf den OK-Button der Ändern-Tabelle werden die Änderungen übernommen.
inline_aendern	In der Haupttabelle erscheinen die änderbaren Felder als Formularfelder. Unter der Haupttabelle gibt es OK- und Abbruch-Button.	Nach Klick auf den OK-Button werden die Änderungen übernommen.
hinzufuegen	Zusätzliche Spalte 'Aktionen'. Spalte enthält in jeder Zeile verlinktes Image 'uebernhemen2.gif'. Ausnahme: In Kombination mit inline_aendern eine Spalte mit Radio-Buttons. Im Spaltentitel befindet sich das Image 'uebernehmen2.gif'.	Klick auf das Image bewirkt das Einfügen des gewählten Tupels in andere DB-Relationen. In Kombination mit inline_aendern muss das entsprechende Radio-Feld ausgewählt werden. Nach Klick auf den OK-Button werden erst die Änderungen durchgeführt, dann erfolgt das Einfügen des gewählten Tupels in andere DB-Relationen.
verschieben	wie bei hinzufuegen	wie bei hinzufuegen. Zusätzlich wird das gewählte Tupel aus der Tabelle gelöscht.
loeschen_multi	Zusätzliche Spalte. Als Spaltenüberschrift erscheint das Image 'loeschen2.gif'. Spalte enthält in jeder Zeile eine Check-Box. Unter der Haupttabelle gibt es OK- und Abbruch-Button.	Nach Klick auf OK-Button werden die mit den Check-Boxen markierten Tupel sofort gelöscht.
hinzufuegen_multi	Zusätzliche Spalte. Als Spaltenüberschrift erscheint das Image 'uebernehmen2.gi'. Spalte enthält in jeder Zeile eine Check-Box. Unter der Haupttabelle gibt es OK- und Abbruch-Button.	Nach Klick auf OK-Button werden die mit den Check-Boxen markierten Tupel in andere DB-Relationen eingefügt. In Kombination mit inline_aendern werden vorher noch Änderungen durchgeführt.
verschieben_multi	wie bei hinzufuegen_multi	wie bei hinzufuegen_multi. Zusätzlich werden die gewählten Tupel aus der Tabelle gelöscht.
neues_tupel	Ausgabe eines Buttons 'Hinzufügen'.	Nach Klick auf den Button wird das Formular für die Eingabe des neuen Datentupels eingeblendet. Nach Klick auf OK-Button des Formulars wird das neue Datentupel eingefügt.

Tabelle 5: Aktionen

- Beispiel für verschieben(_multi): Nach der Web-Anmeldung für eine Vorlesung befinden sich die angemeldeten Studis in einer temporären Tabelle der Datenbank. Aus dieser sollen jetzt die Daten in die Relationen 'student', 'student_besucht_vorlesung' usw. übernommen werden. War die Übernahme erfolgreich kann der Eintrag in der temporären Tabelle gelöscht werden.

Mit dem folgenden Code-Beispiel können neue Daten in die Relation 'student' eingefügt werden. Außerdem ist es möglich Änderungen an bereits eingetragenen Studenten vorzunehmen. Dazu klickt man auf das Ändern-Bildchen und führt die Änderungen in der Ändern-Tabelle durch.

```
$modul_param = array (
  tabellenid => "student",
  leittabelle => "student",
  anzeige => array (
    query => array (
      array (
        SELECT => "vorname, name as nachname,
          matrikelnummer, hauptfach",
      ),
    ),
    aktionen => array(
      aendern => 1,
      neues_tupel => 1,
    ),
  ),
);
modul_tabelle_list($modul_param);
```

4.3.7 Der Eintrag table_param

Über table_param wird für 'anzeige', 'neues_tupel', 'aendern' und 'loeschen' das Aussehen der Tabellen eingestellt. Möglich sind spezielle Spaltenüberschriften, spezielle Formate für die Datenzellen, Spaltenbreiten, Farbeinstellungen etc. Die Details zu table_param können in der Beschreibung zu 'lib-html.php.inc' (siehe 4.6.1) nachgeschlagen werden. Ein Beispiel für ein besonderes Datenzellen-Format findet man im Abschnitt 4.3.5.

4.3.8 Der Eintrag form_param im Detail

Über form_param wird für 'anzeige', 'neues_tupel', 'aendern' und 'loeschen' das Aussehen der Formulare eingestellt. Möglich sind die Verwendung von Select-Feldern, Check-Boxen, die Einstellung der Breiten der Textfelder etc. Die Details zu form_param können in der Beschreibung zu 'lib-html.php.inc' (siehe) nachgeschlagen werden.

Darüber hinaus enthält das Array 'form_param' aber auch Informationen die von 'lib-dbtable' ausgewertet werden. Die für 'lib-dbtable' wesentlichen Bestandteile des Arrays sind mit den Feldnamen indiziert, d.h. mit den Attribut- bzw. Aliasnamen des Query oder bei 'neues_tupel' mit den Schlüsseln des values-Array.

In einer normalisierten Datenbank wird viel mit Fremdschlüsseln und Beziehungstabellen gearbeitet. Beispielsweise wird in den Tabellen, in denen ein Bezug auf den Universitätsnamen notwendig ist nicht der Name selber gespeichert, sondern es wird ein Fremdschlüssel eingefügt, der sich auf eine Universitätstabelle bezieht. Damit in der HTML-Darstellung der Tabelle statt des nichtssagenden Fremdschlüssels, der Name angezeigt wird, gibt es verschiedene Möglichkeiten:

- Man kann die Universitätstabelle mit einen Join in den Query miteinbeziehen.
- Man kann mit Hilfe einer Datenkonvertierungsfunktion (siehe) den Fremdschlüssel vor der HTML-Ausgabe umwandeln. Allerdings sind die Datenkonvertierungsfunktionen eher als

Eintrag	Beschreibung	Default
lookup	SQL-Anfrage, die den Schlüssel und weitere “verständliche“ Werte aus der Datenbank holt.	
type	Legt fest, welcher HTML-Feldtyp im Formular verwendet werden soll. Diese Angabe wird nicht nur bei der Verwendung des Lookup-Konzepts ausgewertet. Im Zusammenhang mit dem Lookup-Konzept machen folgende Typen sinn: select, radio, constant.	Feld änderbar: Bis zu 3 Ergebnistupel: radio; mehr als 3: select Feld nicht änderbar: constant
format	Format-String; damit kann die Darstellung der “verständlichen“ Werte beeinflusst werden.	Alle Werte eines Ergebnistupel, außer dem ersten, werden durch Leerzeichen getrent aneinander gehängt.
null_option	Boolscher Wert, der es ermöglicht zusätzlich zu den aus der Anfrage ermittelten Werten noch einen NULL-Wert anzugeben. Wird dieser Wert ausgewählt wird in der DB an der entsprechenden Stelle NULL eingefügt.	True

Tabelle 6: Bestandteile von $form_param, die von lib-dbtable ausgewertet werden.

statisches Konzept gedacht, bei denen die Konvertierung fest in der Funktion kodiert sind und nicht abhängig von den Daten in der Datenbank ist (z. B. Datums- oder boolsche Formate). Trotzdem könnten mit einer Datenkovertierungsfunktion auch Werte aus der Datenbank verwendet werden.

- Man kann mit lookup-Einträgen in 'form_param' eine kleine SELECT-Anfrage vorgeben, die im wesentlichen den Schlüssel und weitere Werte aus der durch den Fremdschlüssel referenzierten Tabelle ausliest. Statt den Fremdschlüsseln werden dann die Werte aus dieser SELECT-Anfrage ausgegeben.

Der eigentliche Vorteil des Lookup-Konzeptes ergibt sich aber erst bei der Verwendung mit Formularen. Formulare werden dann eingesetzt, wenn die Möglichkeit besteht, Werte zu verändern oder neue Werte einzugeben. Der Anwender weiß in der Regel über die Bedeutung der Fremdschlüssel in einer Tabelle nicht Bescheid. Statt dessen wäre es sinnvoll, dem Anwender eine Liste der möglichen Werte vorzugeben. Diese Liste enthält dann auch nicht die Fremdschlüssel, sondern aussagekräftigere Werte aus der Tabelle, die durch den Fremdschlüssel referenziert wird. Im Falle des Universitätsbeispiels wäre es also wünschenswert, dem Anwender eine Liste der Universitätsnamen vorzugeben, aus der er die gewünschte Universität auswählen kann. Genau das leistet 'lib-dbtable' bei der Verwendung des Lookup-Konzepts. Die angegebene SELECT-Anfrage wird vor der Ausgabe durchgeführt. Dabei wird jeweils der erste Wert der Ergebnistupel als Schlüssel aufgefasst. Die folgenden Werte werden dem Anwender in der Auswahlliste präsentiert. Die Auswahlliste wird in die HTML-Formulare als select-Auswahl oder mit Radio-Buttons eingebaut.

Die für 'lib-dbtable' wesentlichen Bestandteile von 'form_param' sind in Tabelle 6 zusammengefasst.

Bemerkungen zu Tabelle 6:

- lookup: Statt eines Query kann auch ein Array angegeben werden. Beispiel:

```
modul_param => array (
  form_param => array (
    uniname => array (
      lookup => array (
        array ('1', 'LMU'),
```

```
          array ('2', 'TU'),
        ),
      ),
    ),
  );
```

Der jeweils erste Wert wird wie beim Query als Schlüssel aufgefasst.

- type: Den Typ 'constant' gibt es in HTML nicht. Dieser Typ wird verwendet, um auch ein Feld in der Tabelle, das kein Formularfeld ist, konvertieren zu können.

Durch das folgende Beispiel wird eine Tabelle ausgegeben, die anzeigt, welcher Student welche Übungsgruppe besucht. Statt der Gruppennummer wird aus den Tabellen 'uebungsgruppe' und 'wochentag' ein aussagekräftigerer String dargestellt:

```
$modul_param = array (
  tabellenid => "uebungen",
  leittabelle => "student_besucht_uebung",
  required_attribs => array (
    vorlesungnummer => 10,
  ),
  anzeige => array (
    query => array (
      array (
        SELECT => "s.vorname, s.name,
          b.gruppennummer as gruppe",
        FROM => "student_besucht_uebung b,
          student s",
        WHERE => "s.studentnummer=b.studentnummer",
      ),
    ),
    form_param => array (
      gruppe => array (
        type => "constant",
        lookup => "SELECT gruppennummer, tag, zeit
          FROM uebungsgruppe, wochentag
          WHERE vorlesungnummer='10'
            AND wert=wochentag",
        format => "Übung $[gruppennummer]: $[tag] $[zeit]",
      ),
    ),
  ),
);
modul_tabelle_list($modul_param);
```

4.3.9 Der Eintrag appearance im Detail

Wie bereits beschrieben zerfällt ein DB-Table-Submodul in eine Haupttabelle (anzeige) und je nach Aktionsart in die Zwischentabellen aendern und loeschen, sowie in das Formular neues_tupel. Diese vier Bestandteile bestehen wiederum aus einzelnen Elementen. Tabelle 7 listet die Elemente auf und gibt Auskunft darüber, wann die einzelnen Elemente zur Verfügung stehen.

Die Elemente werden, wie das häufig bei der HTML-Programmierung der Fall ist, mit Hilfe einer unsichtbaren Tabelle angeordnet. Die damit möglichen Layoutdetails, sowie weitere Gestaltungsmöglichkeiten können mit dem Array 'appearance' bei Bedarf beeinflusst werden. In der Regel sollten jedoch die Standardeinstellungen ausreichen. Die Einträge 'nur_diese_aktion',

Element	Beschreibung	anzeige	neues_tupel	aendern	loeschen
tabelle	Damit ist die Haupttabelle bzw. die Zwischentabellen oder das Formular für neues_tupel gemeint.	ja	ja	ja	ja
blaettern_a	Navigationsleiste für das seitenweise Blättern.	ja	nein	ja	ja
blaettern_b	Navigationsleiste für das tupelweise Blättern	ja	nein	ja	ja
suchen	Formular zum Suchen, Einschränken und Sortieren	ja	nein	nein	nein
neues_tupel	Button, über den die Aktion 'neues_tupel' ausgelöst werden kann	ja	nein	nein	nein

Tabelle 7: Aus diesen Elementen bestehen 'anzeige', 'neues_tupel', 'aendern' und 'loeschen'

'nur_bei_link_aktion' und 'no_buttons' sind hingegen mehr als reine Layoutparameter. Tabelle 8 zeigt die im Array 'appearance' möglichen Einträge und Tabelle 9 in welchen Submodul-Bestandteilen welche Einträge zur Verfügung stehen.

Im Beispiel werden in der 'anzeige'-Tabelle statt der üblichen 10 jeweils 20 Tupel pro Seite dargestellt. Beim Aufruf der 'aendern'-Tabelle wird die Haupttabelle ausgeblendet:

```
$modul_param = array (
  tabellenid => "student",
  leittabelle => "student",
  anzeige => array (
    query => array (
      array (
        SELECT => "vorname, name as nachname,
          matrikelnummer, hauptfach",
      ),
    ),
    appearance => array (
      zeilen_pro_seite => 20
    ),
    aktionen => array(
      aendern => 1,
    ),
  ),
  aendern => array (
    appearance => array (
      nur_diese_aktion => 1,
    ),
  ),
);
modul_tabelle_list($modul_param);
```

4.3.10 Der Eintrag usertext im Detail

Die vier Bestandteile des DB-Table-Submoduls (anzeige, aendern, loeschen und neues_tupel) lassen sich durch Texte, die in die HTML-Ausgabe eingebaut werden, ergänzen. Das Array 'usertext'

Eintrag	Beschreibung	Default
nur_diese_aktion	Wird eine der Aktionen 'neues_tupel', 'aendern' oder 'loeschen' ausgelöst, wird die zugehörige Zwischentabelle bzw. das Formular zusammen mit der Haupttabelle angezeigt. Setzt man diesen Eintrag auf '1' wird die Haupttabelle nicht angezeigt. Setzt man diesen Eintrag auf '2', werden die anderen DB-Table-Submodule nicht angezeigt (nur bei Verwendung von 'modul_tabelle_list').	'0'
nur_bei_link_aktion	Boolscher Wert, der bei 'true' bewirkt, dass das zugehörige DB-Table-Submodul nur angezeigt wird, wenn in einem DB-Table-Submodul die Aktion 'aendern' oder 'loeschen' ausgelöst wurden (siehe Beschreibung von 'required_attribs' in Abschnitt 4.3.2)	false
zeilen_pro_seite	Integer, der festlegt wieviel Datentupel auf einer Seite jeweils dargestellt werden sollen.	10, bei 'loeschen' 2
reihenfolge	Array, das die Reihenfolge der einzelnen Elemente vorgibt. Hierbei kann für das 'suchen' sogar noch die Reihenfolge der Zeilen für das Suchen und Sortieren verändert werden.	('neues_tupel', 'tabelle', 'blaettern_a', blaettern_b', 'suchen', 'sortieren')
bgcolor	Hintergrundfarbe	'#FFFFFF'
param	Array, das einem table_param-Array (siehe) entspricht, nur das dies für die Layout-Tabelle verwendet wird.	siehe 'libdbtable.inc.php'
abstaende	Array mit den Abstände (in Pixel) jedes einzelnen Elements nach oben. Für den Abstand nach unten gibt es noch den Index 'bottom'. Die Abstände werden mit dem unsichtbaer Image 'blank.gif' realisiert.	(neues_tupel=>2, blaettern_a=>5, blaettern_b=>0, tabelle=>20, suchen=>20, bottom=>80)
neues_tupel_text	String, der vor dem neues_tupel-Buttom erscheint.	siehe HTML-Ausgabe
max_rows_for_mtable	Integer, der bestimmt, ab wieviel Tupel statt einer mtable eine normale Tabelle verwendet werden soll.	2

Tabelle 8: Mögliche Einträge für 'appearance'

Eintrag	anzeige	neues_tupel	aendern	loeschen
nur_diese_aktion	nein	ja	ja	ja
nur_bei_link_aktion	ja	nein	nein	nein
zeilen_pro_seite	ja	nein	ja	ja
reihenfolge	ja	nein	ja	ja
bgcolor	ja	ja	ja	ja
param	ja	ja	ja	ja
abstaende	ja	ja	ja	ja
neues_tupel_text	ja	nein	nein	nein
max_rows_for_mtable	nein	nein	ja	ja

Tabelle 9: Welche Einträge stehen wo zur Verfügung?

Eintrag	Beschreibung
htmlpre	String, der einen Text enthält, der unverändert vor der Tabelle bzw. dem Formular ausgegeben wird.
textpre	String, der einen Text enthält, der nach HTML konvertiert und vor der Tabelle bzw. dem Formular ausgegeben wird.
htmlpost	String, der einen Text enthält, der unverändert nach der Tabelle bzw. dem Formular ausgegeben wird.
textpost	String, der einen Text enthält, der nach HTML konvertiert und nach der Tabelle bzw. dem Formular ausgegeben wird.

Tabelle 10: Die unterschiedlichen Usertexte

unterscheidet vier Textarten (siehe Tabelle 10), die vom Modul-Programmierer z. B. für Beschreibungen zur Benutzung des Moduls benutzt werden können.

Bemerkungen zu Tabelle 10:

- Die HTML-Konvertierung bewirkt, dass Umlaute und Sonderzeichen in die in HTML übliche Codierung umgewandelt werden (z. B. ä = ä).
- Für 'htmlpre' und 'textpre' sind in 'neues_tupel', 'aendern' und 'loeschen' Standardtexte vorgegeben.

Im folgenden Beispiel werden 'htmlpre' und 'textpre' für den 'anzeige'-Teil definiert:

```
$modul_param = array (
  tabellenid => "student",
  leittabelle => "student",
  anzeige => array (
    query => array (
      array (
        SELECT => "vorname, name as nachname,
          matrikelnummer, hauptfach",
      ),
    ),
    usertext => array(
      htmlpre => "<h4>Studenten und ihre Hauptfächer</h4>",
      textpre => "Die nachfolgende Tabelle listet alle
        Studenten und ihren Hauptfächern auf.",
    ),
```

```
  ),
);
modul_tabelle_list($modul_param);
```

4.3.11 Der Eintrag prequeries im Detail

Bevor auf Grund einer Aktion das UPDATE, DELETE oder INSERT in der Datenbank durchgeführt wird, ist es manchmal nützlich einen oder mehrere zusätzliche Queries davorzuschalten. Dadurch können beispielsweise bestimmte Werte zurückgesetzt oder aktualisiert werden. Im Array 'prequeries' kann eine Liste solcher Queries abgelegt werden, wobei die Queries als SQL-Strings vorliegen müssen. In den Queries können auch Platzhalter verwendet werden, die sich auf die Attribute in 'required_attribs' (siehe Abschnitt 4.3.2) beziehen. Zusätzlich sind Platzhalter möglich, die ein Ergebnisattribut eines vorhergegangenen Query aus 'prequeries' referenzieren. Hat der vorhergegangene "Prequery" mehrere Ergebnistupel geliefert, werden für die Ersetzungen der Platzhalter die Werte des ersten Tupels verwendet.

Das folgende Beispiel zeigt eine Tabelle der zu einer Klausur angemeldeten Studenten. Aus der Tabelle können Studenten gelöscht werden. Bevor die Löschung in der DB endgültig durchgeführt wird, werden die in 'prequeries' angegebenen Anfragen durchgeführt. Dabei wird der Feld 'anzahlanmeldungen' in der Relation 'klausur' neu berechnet.

```
$modul_param = array (
  tabellenid => "student_schreibt_klausr",
  leittabelle => "student_schreibt_klausur",
  required_attribs => array(
    vorlesungnummer => 10,
    klausurnummer => 1,
  ),
  anzeige => array (
    aktionen=>array(loeschen=>1),
    query=>array(
      array(
        SELECT=>"s.vorname, s.name, s.matrikelnummer",
        FROM=>"student_schreibt_klausur ssk, student s",
        WHERE=>"s.studentnummer=ssk.studentnummer",
      ),
    ),
  ),
  loeschen => array (
    prequeries => array (
      "SELECT count(studentnummer)-1 as anzahl_neu
        FROM student_schreibt_klausur
        WHERE vorlesungnummer='$(vorlesungnummer)'
          AND klausurnummer='$(klausurnummer)' ",
      "UPDATE klausur
        SET anzahlanmeldungen='$(anzahl_neu)'
        WHERE vorlesungnummer='$(vorlesungnummer)'
          AND klausurnummer='$(klausurnummer)'",
    ),
  ),
);
modul_tabelle_list($modul_param);
```

4.3.12 Der Eintrag composed_attribs im Detail

In Abschnitt 4.3.8 wurde im Zusammenhang mit dem Eintrag 'form_param' das Lookup-Konzept besprochen. Dabei wurde davon ausgegangen, dass der referenzierte Schlüssel immer aus genau einem Attribut besteht. Der Schlüssel muss das erste Attribut in der SQL-Anfrage für den Lookup sein. Besteht der Schlüssel aus mehreren Attributen, müssen diese Attribute ebenfalls am Anfang stehen. Zusätzlich muss diesen Attributen in 'composed_attribs' ein eindeutiger Bezeichner zugeordnet werden, der dann als Index in 'form_param' verwendet werden muss. Der Bezeichner kann auch in anderen Einträgen (z. B. 'cols') benutzt werden.

Im folgenden Beispiel wird eine Liste der Übungsgruppen ausgegeben, wobei die Tutoren der Gruppen verändert werden können. Die Tutoren werden aus der Relation 'arbeitet_fuer_vorlesung' bestimmt, wobei die Referenzierung über den zusammengesetzten Schlüssel ('tutor_istmitarbeiter', 'tutor_nummer') erfolgt. Der Fremdschlüssel erhält als Bezeichner 'betreuer'.

```
$modul_param = array (
  tabellenid => "uebungen",
  leittabelle => "uebungsgruppe",
  required_attribs => array(
    vorlesungnummer => 10,
  ),
  anzeige => array(
    aktionen=>array(inline_aendern=>1),
    query=>array(
      array(
        SELECT=>"uebungsgruppe.*, wochentag.tag",
        FROM=>"uebungsgruppe, wochentag",
        WHERE=>"wochentag.wert=uebungsgruppe.wochentag",
        ORDER=>"gruppennummer",
      ),
    ),
    composed_attribs=>array(
      betreuer=>array("tutor_istmitarbeiter", "tutor_nummer"),
    ),
    cols=>array("gruppennummer", "betreuer", "tag", "zeit"),
    aendern_cols=>array("betreuer"),
    form_param=>array(
      betreuer=>array(
        lookup=>"(SELECT istmitarbeiter, nummer, titel, name, vorname
          FROM arbeitet_fuer_vorlesung a, mitarbeiter m
          WHERE istmitarbeiter<>'0'
            AND a.nummer=m.mitarbeiternummer
            AND vorlesungnummer=10)
          UNION ALL
          (SELECT istmitarbeiter, nummer, 'Stud.' as titel, name, vorname
          FROM arbeitet_fuer_vorlesung a, student s
          WHERE istmitarbeiter='0'
            AND a.nummer=s.studentnummer
            AND vorlesungnummer=10)",
        format=>'$(titel) $(vorname) $(name)',
        type=>"select",
      ),
    ),
  ),
);
modul_tabelle_list($modul_param);
```

Zu beachten ist, dass die Reihenfolge der Attribute in 'composed_attribs' mit der Reihenfolge im zugehörigen Lookup-Teil von 'form_param' übereinstimmt.

4.3.13 Der Eintrag values im Detail

Wenn neue Tupel in eine Relation eingefügt werden, entscheidet in der Regel der Anwendungsbenutzer, welche Werte das neue Tupel enthalten soll. Manchmal kann es aber auch sinnvoll sein, dass bestimmte Attributswerte durch den Modul-Programmierer vorgegeben sind. Diese können im Array 'values' eingetragen werden. Das Array muss mit den Namen der Attribute indiziert werden. Neue Tupel werden durch die Aktionen 'neues_tupel', '(multi_)hinzufuegen' und '(multi_)verschieben' erzeugt, entsprechend steht der Eintrag 'values' in den Bereichen 'neues_tupel' und 'anzeige' von 'modul_param' zur Verfügung.

Bei der Verwendung in 'neues_tupel' werden die in 'values' vorgegebenen Werte im Eingabe-Formular angezeigt. Diese Werte können unter Umständen vom Anwender verändert werden. Der Modul-Programmierer kann im Eintrag 'aendern_cols' (siehe Abschnitt) vorgeben, welche Felder vom Anwender verändert werden dürfen. Bei der Verwendung in 'anzeige' werden die in 'values' vorgegebenen Werte nur intern verarbeitet, der Anwender hat keinen Einfluss darauf.

Die Werte in 'values' können vom Modul-Programmierer fest vorgegeben werden. Alternativ kann er aber auch eine SQL-Anfrage eintragen, die zur Laufzeit ausgewertet wird. Aus dem Ergebnis der Anfrage wird dann der Wert des ersten Attributs im ersten Tupel genommen. Die SQL-Anfrage muss als String eingegeben werden, der mit "SQL:" beginnt. Für 'values' in 'anzeige' steht außerdem die Möglichkeit zur Verfügung, Platzhalter zu verwenden. Diese werden durch Werte aus dem Ergebnis von 'query' ersetzt, wobei immer das letzte in der HTML-Ausgabe angezeigte Tupel zählt.

Das Beispiel erzeugt eine Tabelle der Mitarbeiter des Lehrstuhls. Zusätzlich können neue Mitarbeiter eingefügt werden. Dabei ist der Wert des Attributs 'mitarbeiternummer' (Primärschlüssel der Relation 'mitarbeiter') vom Modul-Programmierer vorgegeben und wird zur Laufzeit aus der SQL-Anfrage bestimmt. Da 'mitarbeiternummer' nicht in aendern_cols auftaucht, kann der Anwender dieses Feld nicht abändern.

```
$modul_param = array (
  tabellenid => "mitarbeiter",
  leittabelle => "mitarbeiter",
  anzeige => array (
    aktionen => array (neues_tupel => 1),
  ),
  neues_tupel => array (
    values=>array(
      mitarbeiternummer=>"SQL:
        SELECT max(mitarbeiternummer)+1 FROM mitarbeiter",
    ),
    aendern_cols=>array( "titel", "name",
      "vorname", "bemerkung", "typ", "firma" ),
  ),
);
modul_tabelle_list($modul_param);
```

Da die Verwendung von 'aendern_cols' in 'neues_tupel' manchmal unnötige Schreibarbeit verursacht, gibt es für 'neues_tupel' eine Variationsmöglichkeit bei der Verwendung von 'values'. Unterhalb von 'values' kann eine zusätzliche Array-Ebene eingeschoben werden, die Schlüssel 'REQ' (required) und 'REC' (recommended) enthalten kann. Die unter 'REC' eingetragenen Werte sind änderbar, die unter 'REQ' nicht. Das folgende Beispiel hat die selbe Wirkung wie obiges:

```
$modul_param = array (
  tabellenid => "mitarbeiter",
```

Bereich	Default	Ersetzung von '*' durch
anzeige	Attribute der Leittabelle ohne Primärschlüssel	cols
aendern	Attribute der Leittabelle ohne Primärschlüssel	cols
neues_tupel	cols	

Tabelle 11: Default-Werte und Ersetzung von '*' in 'aendern_cols'

```
    leittabelle => "mitarbeiter",
    anzeige => array (
      aktionen => array (neues_tupel => 1),
    ),
    neues_tupel => array (
      values=>array(
        REQ=>array(
          mitarbeiternummer=>"SQL:
            SELECT max(mitarbeiternummer)+1 FROM mitarbeiter",
        ),
      ),
    ),
  );
  modul_tabelle_list($modul_param);
```

4.3.14 Der Eintrag aendern_cols im Detail

Mit Hilfe des Eintrags 'aendern_cols' kann der Modul-Programmierer festlegen, welche Felder durch den Anwender verändert werden dürfen. Dieser Eintrag steht in 'aendern' und 'anzeige' (in Zusammenhang mit der Aktion 'inline_aendern') zur Verfügung. Außerdem kann 'aendern_cols' in 'neues_tupel' verwendet werden, wenn der Programmierer verhindern will, dass der Anwender vorgegebene Werte (siehe Abschnitt 4.3.13) verändern kann. Wie der Eintrag 'cols' (siehe Abschnitt 4.3.5) ist auch 'aendern_cols' ein Array, das eine Liste der Aliasnamen der Felder aus dem Query bzw. aus 'values' enthält. Außerdem kann 'aendern_cols' einen Stern ('*') als Wildcard enthalten. Default-Werte und Bedeutung von '*' zeigt Tabelle 11. Ein Beispiel für die Verwendung von 'aendern_cols' findet sich in Abschnitt 4.3.13.

4.3.15 Der Eintrag wrapper im Detail

Der Eintrag 'wrapper' kann Bestandteil der Bereiche 'neues_tupel', 'anzeige' und 'loeschen' sein. Wird über die Aktion 'neues_tupel' ein neues Datentupel erzeugt, so wird es standardmäßig in die Leittabelle eingefügt. Manchmal ist es aber notwendig, dass gleichzeitig Einfügungen in andere Relationen vorgenommen werden, um beispielsweise referentielle Integritäten zu erfüllen. Zu diesem Zweck steht das Array 'wrapper' (hier Bestandteil von 'neues_tupel') zur Verfügung, das nach den Namen der Relationen indiziert ist, in die Einfügungen vorgenommen werden sollen. Der zugehörige Wert ist wiederum ein Array, das nun nach den Attributnamen der Relation indiziert ist. Der den Attributen zugeordnete Wert kann ein Platzhalter sein, der sich auf ein Feld des neues_tupel-Formulars bezieht und über 'values' (siehe Abschnitt 4.3.13) definiert ist. Als Platzhalter sollte die Version mit den runden Klammern ('$(...)') verwendet werden, damit keine Konvertierungen durchgeführt werden (siehe Abschnitt). An Stelle der Platzhalter können auch SQL-Anfragen, in der Form wie sie bereits in 'values' Verwendung gefunden haben (als String mit führendem 'SQL:'), eingesetzt werden. Durch 'wrapper' ist auf diese Weise also festgelegt, welche Werte aus dem neues_tupel-Formular, welchen Attributen in welchen Relationen zugeordnet werden müssen. Werden im Formular auch Felder verwendet, die keine Entsprechung in der Leittabelle haben, so muss auch für die Leittabelle ein wrapper angegeben werden.

Neben der Aktion 'neues_tupel' wird 'wrapper' auch bei den Aktionen 'hinzufuegen(_multi)' und 'verschieben(_multi)' (hier Bestandteil von 'anzeige') benötigt. Hier muss 'wrapper' definiert sein, damit die Anwendung weiß, in welche Relationen die Einfügungen vorgenommen werden sollen. Im Gegensatz zu 'neues_tupel' beziehen sich die Platzhalter hier auch auf die Ergebnismenge des Query.

Im Zusammenhang mit der Aktion 'loeschen' kann 'wrapper' (hier Bestandteil von 'loeschen') verwendet werden, um gleichzeitig Löschungen in weiteren Relationen als der Leittabelle durchzuführen. Das Format ist identisch zu oben. Dabei sind nur Attribute zu spezifizieren, mit denen der zu löschenden Eintrag in der Relation eindeutig identifiziert werden kann. Die Platzhalter beziehen sich auch hier auf die Ergebnisse des Query. Das Löschen in der Leittabelle wird in der Regel nach den Primärschlüsseln vorgenommen. Soll nach anderen Attributen in der Leittabelle gelöscht werden, kann dies ebenfalls durch einen entsprechenden Wrapper-Eintrag vorgenommen werden.

Das folgende Beispiel erzeugt in der HTML-Ausgabe zwei Tabellen. Die obere stellt die für Vorlesungen reservierten Räume dar, die untere zeigt alle Räume. Aus der unteren können durch Anklicken Räume zur Relation 'vorlesung_findet_statt_in' hinzugefügt werden, die dann auch in der oberen Tabelle angezeigt werden. Der Wrapper ordnet das Feld 'raumnummer' dem Attribut 'raumnummer' der Relation 'vorlesung_findet_statt_in' zu. Darüber werden noch einige feste Werte vorgegeben (wichtig ist die richtige Vorlesungsnummer).

```
$modul_param1 = array (
  tabellenid => "vorlesungsräume",
  leittabelle => "vorlesung_findet_statt_in",
  anzeige => array (
    query=>array(
      array(
        SELECT=>"raum.name as raum, raum.sitzplaetze,
          raum.ausstattung, vr.*",
        FROM=>"vorlesung_findet_statt_in vr, raum",
        WHERE=>"vr.raumnummer=raum.raumnummer",
      ),
    ),
  ),
);
$modul_param2 = array (
  tabellenid => "räume",
  leittabelle => "raum",
  anzeige => array(
    aktionen=>array(hinzufuegen=>1),
    wrapper=>array(
      vorlesung_findet_statt_in=>array(
        raumnummer=>'$(raumnummer)',
        wochentag=>'1',
        zeit=>wftime_unix2sql(time(), "time"),
        minutendauer=>'120',
        vorlesungnummer=>'10',
      ),
    ),
  ),
);
modul_tabelle_list($modul_param1, $modul_param2);
```

4.3.16 Der Eintrag wrap_param im Detail

Der Eintrag 'wrap_param' (Typ: Array) steht in den Abschnitten 'anzeige' und 'neues_tupel' des Arrays 'modul_param' zur Verfügung. Er dient dazu das Verhalten der Anwendung bei geschei-

Eintrag	Verhalten der Anwendung nach fehlgeschlagenem INSERT
update_on_insert_error	Unter Umständen existiert bereits ein Tupel mit dem selben Primärschlüssel. Dann kann es sinnvoll sein ein UPDATE zu versuchen.
dont_add_primkeys	Im Zusammenhang mit 'update_on_insert_error'. Das zu ändernde Tupel der aktuellen Relation wird normalerweise über seinen Primärschlüssel bestimmt. Es kann aber auch über den Primärschlüssel der Leittabelle bestimmt werden, wenn dieser Eintrag verwendet wird.
delete_after_insert_error	Das Tupel wird bei den Aktionen 'verschieben' und 'verschieben_multi' trotz fehgeschlagenem INSERT aus der Leittabelle gelöscht.
go_back_on_insert_error	Nach einem fehlgeschlagenem INSERT wird abgebrochen und es wird versucht, die bisherigen INSERTs wieder rückgängig zu machen.

Tabelle 12: Die Einträge von 'wrap_param', die das Verhalten der Anwendung bei einem fehlgeschlagenen INSERT beeinflussen

terten Einfügeoperationen in eine Relation zu beeinflussen. Standardmäßig macht die Anwendung nach einem gescheiterten INSERT mit den anderen Einfügeoperationen, die durch 'wrapper' (siehe Abschnitt 4.3.15) vorgegeben sind, weiter. Im Zusammenhang mit den Aktionen 'verschieben' und 'verschieben_multi' wird bei einem fehlgeschlagenen INSERT das Tupel nicht aus der Leittabelle gelöscht. Tabelle 12 zeigt, welche Einträge 'wrap_param' enthalten kann und was für einen Einfluss diese auf das Verhalten der Anwendung haben. Den Einträgen wird als Wert immer eine Liste von Tabellennamen übergeben, für die das entsprechende Verhalten gelten soll.

Aus dem Rahmen fällt der Eintrag all_attribs (nur bei 'anzeige'). Er beeinflusst nicht das Verhalten nach einem fehlgeschlagenem INSERT, sondern stellt nur eine Schreiberleichterung dar. Für die darin aufgelisteten Tabellen wird der zugehörige Teil von 'wrapper' ergänzt oder erstellt. Das sieht so aus, dass für alle Attribute der jeweiligen Tabelle ein Wrapper-Eintrag automatisch erzeugt wird.

Im folgenden ist ein Ausschnitt aus 'modul-Vorlesungsanmeldung.inc.php' dargestellt. Der Ausschnitt gehört zu einer Tabelle, die die Anmeldungen der Studenten anzeigt. Diese Anmeldungen befinden sich in einer temporären Relation des öffentlichen Bereichs der DB. Der Anwender kann aus der dargestellten Tabelle Studis auswählen, die dann in die Relationen 'student', 'student_besucht_vorlesung', 'student_besucht_uebung' und 'mailadressestudent' des geschützten Bereichs der DB übernommen werden.

```
...
wrap_param=>array(
  all_attribs=>array("student"),
  update_on_insert_error=>array(
    "student",
    "mailadressenstudent"
  ),
  delete_after_insert_error=>array(
    "student_besucht_vorlesung",
    "student_besucht_uebung"
  ),
  dont_add_primkeys=>array("student"),
),
```

```
wrapper=>array(
  student=>array(
    studentnummer=>"$(studentnummer)",
    universitaetsnummer=>"$(uni)",
  ),
  student_besucht_vorlesung=>array(
    vorlesungnummer=>$vn_nr,
    studentnummer=>"$(studentnummer)",
  ),
  student_besucht_uebung=>array(
    vorlesungnummer=>$vn_nr,
    studentnummer=>"$(studentnummer)",
    besuchtuebung=>"$(besuchtuebung)",
    gruppennummerpraeferenz1=>"$(gruppennummerpraeferenz1)",
    gruppennummerpraeferenz2=>"$(gruppennummerpraeferenz2)",
    gruppennummerpraeferenz3=>"$(gruppennummerpraeferenz3)",
  ),
  mailadressenstudent=>array(
    studentnummer=>"$(studentnummer)",
    mailadresse=>"$(mailadresse)",
    hauptadresse=>1,
  ),
),
...
```

4.3.17 Der Eintrag dataconversion im Detail

Oft ist das DB-interne Format eines bestimmten Attributs nicht für die Präsentation im Browser geeignet. Dies trifft beispielsweise für Datums- und Zeitformate zu. Auf der anderen Seite ist es möglich, dass eine Benutzereingabe nicht zum DB-internen Format passt. Mit Hilfe der Funktionen in 'lib-dataconv.inc.php' können in beiden Richtungen entsprechende Umwandlungen der Daten vorgenommen werden. Welche Funktionen für die Umwandlung der Werte bestimmter Attribute zu verwenden sind, kann mit Hilfe des Eintrags 'dataconversion' angegeben werden. 'dataconversion' ist ein Array, das mit den für die Konvertierung vorgesehenen Attributen indiziert ist. Als Werte enthält das Array entweder direkt den Namen der Datenkonvertierungs-Funktion oder ein Array das multiple Konvertierungen ermöglicht.

Die Tabelle im Beispiel gibt die Klausuren Zeit aus. 'dataconv_datetime' bewirkt, dass nur Datum und Uhrzeit ausgegeben werden. Die Zeitzone wird nicht ausgegeben. Es können auch neue Klausuren eingetragen werden. Die Nummer der Klausur wird dabei automatisch mit der SQL-Anfrage in 'values' bestimmt. Wenn noch keine Klausur vorhanden ist liefert die SQL-Anfrage nichts zurück. In diesem Fall greift die Datenkonvertierungsfunktion 'dataconv_null_1', die aus 'nichts' eine '1' macht.

```
$modul_param = array (
  tabellenid => "klausuren",
  leittabelle => "klausur",
  required_attribs => array (
    vorlesungnummer => 10,
  ),
  anzeige => array (
    aktionen => array (
      neues_tupel => 1,
      inline_aendern => 1,
    ),
```

```
    cols => array ("klausurnummer", "zeit"),
    dataconversion=>array(
      zeit=>"dataconv_datetime"
    ),
  ),
  neues_tupel => array (
    cols => array ("klausurnummer", "zeit"),
    aendern_cols => array ("zeit"),
    values=>array(
      klausurnummer=>"SQL: SELECT max(klausurnummer)+1
        FROM klausur
        WHERE vorlesungnummer='10'"
    ),
    dataconversion=>array(
      klausurnummer=>"dataconv_null_1"
    ),
  ),
);
modul_tabelle_list($modul_param);
```

4.3.18 Der Eintrag tabelle im Detail

In manchen Situationen im Zusammenhang mit der Aktion 'aendern' kann es günstig sein, eine andere Tabelle als die Leittabelle als Basis für die Änderungen zu verwenden. Die Tabelle im folgenden Beispiel stellt alle Teilnehmer der Vorlesung mit der Nummer 10 dar. Klickt man auf Ändern wird für den ausgewählten Student aufgeschlüsselt, wieviel Punkte er auf die einzelnen Aufgaben der Übungen und Klausuren erhalten hat. Als Leittabelle wird 'student_besucht_vorlesung' genommen, da darin alle Studenten der Vorlesung erfasst sind, auch die die keine Punkte erhalten haben. Für die Änderungen wird die Tabelle 'student_erhaelt_punkte' herangezogen. Wird in 'aendern' der Eintrag 'tabelle' verwendet, muss auch ein 'query' explizit angegeben werden.

```
$modul_param = array (
  tabellenid => "punkte",
  leittabelle => "student_besucht_vorlesung",
  required_attribs => array (
    vorlesungnummer => 10,
  ),
  anzeige => array (
    aktionen => array ( aendern => 1, ),
    query => array (
      array (
        SELECT => "s.vorname, s.name, s.matrikelnummer",
        FROM => "student_besucht_vorlesung v, student s",
        WHERE => "v.studentnummer=s.studentnummer",
      ),
    ),
  ),
  aendern => array (
    tabelle => "student_erhaelt_punkte",
    query => array (
      array (
        SELECT => "studentnummer, aufgabennummer, erreichtepunkte",
        FROM => "student_erhaelt_punkte",
      ),
```

```
      ),
    ),
  );
  modul_tabelle_list($modul_param);
```

4.3.19 Der Eintrag ziellink im Detail

Normalerweise wird mit jedem Link und mit jedem Button wieder das Skript 'workflow.php' aufgerufen. Dies ist auch notwendig, da die Steuerung der gesamten Anwendung durch dieses Skript erfolgt. Es wird deshalb kaum sinnvoll sein, in den Modulen eine andere Seite als Ziel für die Links anzugeben. Sollte es dennoch mal nötig sein, kann dies über den Eintrag 'ziellink' geschehen.

4.4 Submodule für die Steuerung externer Module

4.4.1 Einleitung

Es gibt verschiedenen Situationen, in denen ein Zugriff auf Teilbereiche der Datenbank auch von außerhalb der Workflowanwendung nötig ist. Beispiele sind die Anmeldung zu einer Vorlesung oder die Korrektur einer Klausur. Dabei sollten für die Studenten bzw. Tutoren nur die Bereiche der Datenbank zugänglich sein, die für diese Tätigkeiten notwendig sind. Dies wird durch Views oder temporäre Tabellen und entsprechende Zugriffsrechte darauf erreicht. Das Erzeugen und Löschen der Views bzw. temporären Tabellen, einschließlich Zuweisung und Entzug der Rechte, kann aus der Workflowanwendung heraus gesteuert werden.

Die Namen der erzeugten Views bzw. temporären Tabellen werden in der Datenbank in der Relation workflow_extmod_param gespeichert. Diese können dann von einem externen Modul, das z. B. für die Anmeldung zu einer Vorlesung geschrieben wurde, ausgelesen werden. Unter einem externen Modul wird ein PHP-Skript verstanden, das unabhängig von der Workflowanwendung existiert, aber trotzdem die Libraries und Konzepte der Anwendung benutzt. Natürlich kann auch jede andere Anwendung auf die Relation zugreifen. Damit das Auslesen der Werte gelingt, muss ein allgemeines Leserecht auf die Relation workflow_extmod_param existieren. Bei der Erzeugung der Views und temporären Tabellen können neben deren Namen auch noch andere Werte über die Relation an das externe Modul übergeben werden. Beim Löschen der Views wird die Relation von allen diesen Einträgen gereinigt.

Wie bei den anderen Submodulen steht auch hier eine Funktion zur Verfügung, der man in einem Parameter-Array alle nötigen Informationen übergibt, und die das übrige erledigt. Diese Funktion heißt hier "modul_extmod ($modul_param)" und wird von der Library lib-extmodul-control.inc.php zur Verfügung gestellt. Die von der Funktion erzeugte HTML-Ausgabe ist nicht besonders umfangreich und besteht nur aus Buttons. Wird eine Workflow-Seite mit einem solchen Modul zum ersten Mal aufgerufen, erscheint ein Button mit der Beschriftung "Freischalten". Nach Betätigen des Knopfes werden die Views und temporären Tabellen erzeugt, die Rechte werden gesetzt und die Einträge in workflow_extmod_param vorgenommen. Daraufhin erscheint ein Knopf "Sperren", mit dessen Hilfe die Views, die Rechte und die Einträge in der Relation wieder gelöscht werden können. Die temporäre Tabelle, sofern eine angelegt wurde, bleibt aber erhalten, damit sie später vom Workflow-Anwender bearbeitet werden kann. Daraufhin erscheint wieder der Freischalten-Button und zusätzlich ein Löschen-Button, wenn eine temporäre Tabelle existiert. Diese wird erst nach Benutzen des Buttons gelöscht.

Die Funktion modul_extmod stützt sich neben den zusätzlich in lib-extmodul-control.inc.php enthaltenen Funktionen im wesentlichen auf Funktionen aus folgenden Bibliotheken:

- lib-view-control.inc.php: In dieser Bibliothek befinden sich alle Funktionen, die zum Erzeugen und Löschen von Views nötig sind. Dazu stehen die Funktionen view_create bzw. view_drop zur Verfügung. Je nach verwendeter Datenbank rufen diese Funktionen die zur Datenbank passenden Routinen auf. Unterstützt werden Interbase, PostgreSQL und MySQL. Letztere hat den Nachteil, dass sie keine Views kennt, und diese deshalb mit temporären Tabellen simuliert werden müssen. Von Interesse ist noch die Funktion view_info, die den Namen und andere Informationen eines Views zurückliefert.

- lib-tmptable.inc.php: Entsprechend zu oben, gibt es in dieser Bibliothek Funktionen tmptable_create und tmptable_drop zum Erzeugen bzw. Löschen von temporären Tabellen. Informationen zu einer temporären Tabelle werden wie oben mit der Funktion view_info ausgelesen.

4.4.2 Zusammensetzung des Parameter-Array $modul_param

Im Parameter-Array $modul_param für die Funktion modul_extmod sind alle Informationen enthalten, die für die Steuerung des Submoduls selber und für Steuerung des Zugriffs auf die Datenbank von außen nötig sind. Diese Bestandteile des Arrays sind in Tabelle 13 zusammengefasst.

Zur Tabelle sind noch folgende Bemerkungen zu machen:

- extmod_id, aufruf_id: Die Relation workflow_extmod_param enthält Einträge für die verschiedenen externen Module, auch aus unterschiedlichen Workflows. Damit die Einträge den richtigen externen Modulen zugeordnet werden können, wird in der Relation immer die Workflownummer und die extmod_id mit abgespeichert. Unter Umständen wird ein externes Modul mehrmals in einem Workflow benutzt. Beispielsweise kann ein Anmeldungs-Modul für eine Vorlesungs- und eine Klausuranmeldung verwendet werden. Damit klar ist für welchen Verwendungszweck die Werte in workflow_extmod_param bestimmt sind, wird eine Aufruf-ID eingetragen. Über die Attribute workflownummer, extmod_id und aufruf_id kann das externe Modul so "seine" Parameter aus der Relation heraus holen.

- user: Sind in insert, check oder new_tupel Benutzer definiert, wird der Eintrag in 'user' für den zugehörigen View bzw. temporäre Tabelle ignoriert. Der String 'user' kann deshalb als eine Art Default aufgefasst werden. Gruppen werden durch ein vor dem Gruppennamen stehendes "group" gekennzeichnet.

Darüberhinaus können wie eingangs erwähnt beliebige Werte über die Relation workflow_extmod_param an das externe Modul übergeben werden. Die zugehörigen Schlüssel im Array beginnen alle mit "ext_" und sind in Tabelle 14 beschrieben. Das "ext_" wird vor dem Eintrag in die Relation workflow_extmod_param entfernt. Die Schlüssel beziehen sich fast alle auf die aus DBTable-Submodul (Abschnitt 4.3) bekannten Schlüssel. Ihre genaue Verwendung sollte man deshalb dort nachschlagen. Allerdings kann der Modul-Programmierer die Schlüssel auch für andere Zwecke "missbrauchen".

Zur Tabelle sind noch folgenden Bemerkungen nötig:

- ext_texte: Beginnt der Schlüssel eines Textes mit "html" wird der Text vor dem Eintrag in workflow_extmod_param für HTML vorbereitet (z. B. Umlautkodierung: ä -> ä). Als Schlüssel können ansonsten beliebige Strings verwendet werden, beispielsweise für Titel, Überschriften und Meldungen.

- ext_form_param: Lookup-Queries werden vor dem Eintrag in workflow_extmod_param aufgelöst, da davon ausgegangen werden muss, dass dies aus dem externen Modul heraus wegen fehlender DB-Rechte nicht möglich ist.

Das folgende Beispiel erzeugt noch keine Views oder temporäre Tabellen. Es werden lediglich die Texte in ext_texte in die Relation workflow_extmod_param eingetragen. Dabei wird der Text mit dem Schlüssel "htmltext" für HTML kodiert. Die Einträge in die Relation sind für ein externes Modul mit der ID "extmod1" unter einem Aufruf mit der ID "test" bestimmt.

```
$modul_param = array(
  id=>"steuerung_extmod",
  extmod_id=>"extmod1",
```

Schlüssel	zugehöriger Wert des Attributs 'paramkey' in der Relation 'workflow_extmod_param'	Beschreibung
id		String, mandatory. Wie alle anderen Submodule, muss auch diesem Submodul eine im Workflow-Schritt eindeutige ID zugewiesen werden, damit die Ablaufsteuerung funktioniert.
extmod_id		String, mandatory. Über diese ID wird festgelegt, für welches externe Modul die Werte bestimmt sind, die in die Relation 'workflow_extmod_param' eingetragen werden. Das externe Modul kann mit Hilfe dieser ID seine Werte auslesen.
aufruf_id		String, mandatory. Über diese ID wird festgelegt, für welchen Aufruf des externen Moduls die Werte bestimmt sind, die in die Relation 'workflow_extmod_param' eingetragen werden. Das externe Modul kann mit Hilfe dieser ID die zur Aufrufsituation passenden Werte auslesen.
user		String. Eine durch Kommata getrennte Liste von Benutzern, die Zugriff auf die Views und temporären Tabellen erhalten sollen.
insert	table_insert_list und table_get_attribs	Array. Enthält Informationen zu einem oder mehreren Views. Diese Views können im externen Modul zum Einfügen oder Aktualisieren von Tupeln verwendet werden.
check	table_check	Array. Wie 'insert', aber kann nur Informationen zu genau einem View aufnehmen. Dieser View kann im externen Modul zur Überprüfung von eingegebenen Werten verwendet werden.
new_tupel	table_new	Array. Enthält alle Informationen zur Erzeugung einer temporären Tabelle. Diese kann im externen Modul verwendet werden um neue Tupel aufzunehmen.
usertext		Array. Texte, die am Anfang und Ende des Submoduls ausgegeben werden. Siehe

Tabelle 13: Steuerungsinformationen im Array $modul_param

Schlüssel	Beschreibung
ext_texte	Array. Enthält Texte, die im externen Modul ausgegeben werden sollen. Dies ist besonders dann sinnvoll, wenn ein Modul in verschiedenen Situationen verwendet werden soll.
ext_form_param	Array. Parameter für Formulare.
ext_cols	Array. Liste von Feldern, die angezeigt werden sollen. Für die Views gedacht.
ext_tmp_table_cols	Array. Liste von Feldern, die angezeigt werden sollen. Für die temporäre Tabelle gedacht.
ext_table_param	Array. Parameter für Tabellen
ext_values	Array. Durch den Modulu-Programmierer vorgegebene Werte
ext_dataconversion	Array. Datenkonvertierungsfunktionen, die auf die angegebenen Felder angewendet werden sollen.

Tabelle 14: Zusätzliche Informationen an das externe Modul

```
    aufruf_id=>"test",
    usertext=>array(
      htmlpre=>"<h4>Eintragen und Löschen von
        Tupeln in workflow_extmod_param</h4>",
    ),
    ext_texte=>array(
      titel=>"Externes Modul",
      htmltext=>"Möglicher Aufruf des externen Moduls",
    ),
);
modul_extmod($modul_param);
```

4.4.3 Übergeben von Werten an ein externes Modul über die Relation workflow_extmod_param

Über die Relation workflow_extmod_param können Variablen an ein externes Modul/Skript übergeben werden. Die Relation verwendet als Schlüssel die Attribute 'workflownummer', 'extmod_id', 'aufruf_id' und 'paramnr'. Letzteres hat nur technischen Charakter. Der Variablenname wird im Attribut 'paramkey', der Variablenwert in 'paramvalue' abgelegt. Es können auch Arrays über 'workflow_extmod_param' übergeben werden. Diese werden jedoch in ihre einzelnen Bestandteile zerlegt und so in die Relation eingetragen. Ein zweidimensionales Array beispielsweise verursacht mehrere Tupel, wobei die Einträge unter 'paramkey' die folgende Form haben:

```
arrayname[key1][key2]
```

Für das Auslesen von Tupeln aus der Relation 'workflow_extmod_param' steht die Funktion 'extmod_param(wf_nr, extmod_id, aufruf_id)' in 'lib-extmodul.inc.php' zur Verfügung. Da sich in der Relation die Informationen zu allen aktuell freigeschalteten externen Modulen befinden, müssen die richtigen mit Hilfe der übergebenen Parameter 'wf_nr', 'extmod_id' und 'aufruf_id' selektiert werden. Die Funktion gibt ein Array mit den ausgewählten Werten zurück, wobei es mit den Namen der Variablen indiziert ist. Falls über die Relation auch Arrays übergeben worden sind, müssen diese wieder zusammengebaut werden. Dabei kann die Funktion 'modparam_getarray($param, $arrayname)' aus 'lib-modul_param.inc.php' helfen. In '$param' wird der Funktion das Ergebnis des 'extmod_param'-Aufrufs, in '$arrayname' der Name des Arrays übergeben. Die Funktion liefert als Resultat das zusammengesetzte Array. Ein Wort noch zur Library 'lib_extmodul.inc.php'. Sie ist speziell zur Aufnahme von Funktionen und Daten gedacht, die in externen Modulen benötigt werden.

Wie in Abschnitt 4.4.2 beschrieben, trägt die Funktion 'modul_extmod' die Namen der Views und temporären Tabellen, sowie mit den "ext_"-Schlüsseln weiter Informationen für die externen Module in die Relation 'workflow_extmod_param' ein. Sollte dies nicht ausreichen, können aus einem ExtMod-Submodul heraus auch direkt Einträge in die Relation vorgenommen werde, indem die Funktionen 'wf_extmod_param_set' und 'wf_extmod_param_setarray' aus 'lib-extmodulcontrol.inc.php' verwendet werden. Erstere ist für einfache Variablen, letztere für Arrays. Der Funktion 'wf_extmod_param_set' müssen die Parameter (in dieser Reihenfolge) 'extmod_id', 'aufruf_id', 'variablenname' und 'variablenwert' übergeben werden. Bei 'wf_extmod_param_setarray' sind die Parameter (in dieser Reihenfolge) 'zu_speicherndes_Array', 'arrayname', 'extmod_id' und 'aufruf_id'.

Im folgenden Beispiel wird ein Array mit Queries ('query_extern') mit dem ersten Funktionsaufruf in die DB eingetragen, wobei als Array-Name 'queries' verwendet wird. Anschließend werden alle zum externen Modul gehörenden Werte aus der DB wieder ausgelesen und das Array 'queries' rekonstruiert.

```
wf_extmod_param_setarray($query_extern, ``queries'', ``extmod1'', ``test'');
$param = extmod_param(10, ``extmod1'', ``test'');
$queries = modparam_getarray($param, ``queries'');
```

4.4.4 Die Einträge 'insert' und 'check' im Detail

Im Parameter-Array für die Funktion 'modul_extmod' gibt es die Schlüssel 'insert' und 'check'. Unter ihnen werden die Details übergeben, die zur Erzeugung von Views notwendig sind. In 'insert' können mehrere Views, in 'check' nur ein View definiert werden. Die Insert-Views sind für neue Tupel gedacht, deren Daten beispielsweise über ein HTML-Formular eines externen Moduls/Skripts eingegeben wurden. Die Namen der Views werden durch Komma getrennt in einem String zusammengefasst und unter dem Variablennamen table_insert_list in der Relation 'workflow_extmod_param' gespeichert. Für die Insert-Views muss festgelegt werden, welche der eingegebenen Werte in welche Views gelangen sollen. Dazu dient ein String, der unter dem Schlüssel 'get_attrib' für jeden Insert-View abgelegt werden muss. Der String enthält durch Komma getrennt eine Liste der Feldnamen. Da unter Umständen die Attributnamen des Views von den Namen der eingegeben Werte abweichen, kann im String die Notation 'attributname:wertname' benutzt werden. Die Strings werden in der DB unter dem Namen 'table_get_attrib[viewname]' abgelegt. Für jeden View enthält die DB einen solchen String.

Der Check-View ist geeignet, um eingegebene Werte mit Daten in der Datenbank zu vergleichen. Der Name des Views wird in der Relation 'workflow_extmod_param' unter 'table_check' gespeichert. In der Tabelle 15 sind die Bestandteile des 'insert'- und 'check'-Array zusammengefasst.

Bemerkungen:

- 'viewid': Die Generierung der Viewnamen durch die Workflowanwendung soll sicherstellen, dass die Namen in der DB eindeutig sind. Die View-IDs müssen hingegen nur für einen Workflow eindeutig sein. Dadurch kann ein Modul mit einem ExtMod-Submodul in mehreren Workflows verwendet werden, ohne dass besondere Vorkehrungen für die Viewnamen getroffen werden müssen. Die Viewnamen werden in die Relation 'workflow_extmod_param' eingetragen. Die Viewnamen können außerdem mit Hilfe der Funktion 'view_info($viewid)' abgefragt werden. Die Funktion gibt ein Array zurück, das unter dem Schlüssel 'name' den Viewnamen enthält.

- 'query': Das Query-Array kann neben den in Abschnitt 4.3.4 beschriebenen Einträgen für das Datenbank-Management-System MySQL außerdem einen Schlüssel

Schlüssel	insert	check	Beschreibung
viewid	ja	ja	String, mandatory. Mit Hilfe dieser ID werden die Views identifiziert. Die IDs müssen innerhalb eines Workflows eindeutig sein. Die eigentlichen Namen der Views werden von der Anwendung generiert und haben die Form: view<aktuelle_zeit>.
query	ja	ja	Array, mandatory. Das Array hat die in Abschnitt beschriebene Struktur. Mit diesem Query wird der View definiert.
leittabelle	ja	ja	String, mandatory. Name der Leittabelle, auf die sich der View beziehen soll. Zwar können im View durch Joins mehrere Tabellen der Datenbank zusammengefasst werden, Updates und Inserts haben jedoch nur Auswirkung auf die Leittabelle.
rechte	ja	ja	String oder Array. Die hier definierten Rechte werden für den View gesetzt. Bei der ersten Variante werden die Rechte für den unter 'user' (siehe Abschnitt 4.3.2) eingetragenen Benutzer gesetzt. Bei der Array-Variante werden die Benutzernamen als Schlüssel und die zugeordneten Rechte als Werte eingetragen. Die Rechte sind in beiden Varianten durch Komma getrennt.
req	ja	nein	Array. Das Array hat als Schlüssel Attributnamen der Leittabelle und dazu passende Werte. Wird durch einen View ein Tupel in die Leittabelle neu eingefügt, werden für die entsprechenden Attribute diese Werte verwendet. Dadurch kann beispielsweise sichergestellt werden, dass nur neue Tupel für den aktuellen Workflow erstellt werden.
get_attrib	ja	nein	String. Durch Komma getrennte Liste von Attributnamen des Views. Beschreibung siehe Text.

Tabelle 15: Bestandteile des 'insert'- und 'check'-Array

'WITH_CHECK_OPTION' für beschreibbare Views enthalten. Bei einem INSERT oder UPDATE wird dann überprüft, ob sie den in der View-Definition enthaltenen Bedingungen genügen. Wenn beispielsweise der View nur Tupel enthält, bei denen das Attribut xyz größer ist als 10, dann können durch den View keine Tupel eingefügt werden, bei denen xyz kleiner ist als 10.
Der Schlüssel 'GROUP' sollte nicht benutzt werden, da die Views, damit nicht vollständig umgehen können.

- 'leittabelle': Da die untersuchten DBMS gar nicht oder nur eingeschränkt Write-Through-Views unterstützen, wird das Durchschreiben durch den View von der Workflow-Anwendung nachgebildet. Da dies eine sehr komplexe Angelegenheit ist, wird das Durchschreiben auf die angegebene Leittabelle beschränkt.
- 'req': Dieser Schlüssel wird bei der Verwendung von MySQL als DBMS nicht berücksichtigt.

Im folgendem Beispiel wird von diesem Szenario ausgegangen: Über ein Web-Formular kann sich ein Student über seine Matrikelnummer und den Uniname für eine Vorlesung (Vorlesungsnummer = 10) anmelden. Dabei muss er außerdem auswählen, ob und an welchen Übungen er teilnehmen möchte. Dazu sind zwei Insert-Views nötig. Über den ersten wird die Anmeldung in die Relation 'student_besucht_vorlesung', über den zweiten in die Relation 'student_besucht_uebung' übernommen. Über den Check-View wird sichergestellt, dass der Student in der DB existiert. Außerdem wird über den Check-View der Primärschlüssel 'studentnummer' bestimmt, der in 'student_besucht_vorlesung' und 'student_besucht_uebung' als Fremdschlüssel gebraucht wird.

```
$modul_param = array(
  id=>"steuerung_extmod",
  extmod_id=>"extmod1",
  aufruf_id=>"test",
  insert=>array(
    array(
      viewid=>"vorlesungsanmeldung_insert",
      leittabelle=>"student_besucht_vorlesung",
      query=>array(
        array(
          SELECT=>"v.vorlesungnummer, v.studentnummer",
          FROM=>"student_besucht_vorlesung v",
          WHERE=>"v.vorlesungnummer='10'",
          WITH_CHECK_OPTION=>1
        ),
      ),
      req=>array(
        vorlesungnummer=>10,
      ),
      rechte=>"SELECT, INSERT",
    ),
    array(
      viewid=>"uebungsanmeldung_insert",
      leittabelle=>"student_besucht_uebung",
      query=>array(
        array(
          SELECT=>"u.vorlesungnummer, u.studentnummer, u.besuchtuebung,
            u.gruppennummerpraeferenz1, u.gruppennummerpraeferenz2",
          FROM=>"student_besucht_uebung u",
          WHERE=>"u.vorlesungnummer='10'",
          WITH_CHECK_OPTION=>1
```

```
            ),
          ),
          get_attrib=>"besuchtuebung, gruppennummerpraeferenz1,
            gruppennummerpraeferenz2",
          req=>array(
            vorlesungnummer=>10,
          ),
          rechte=>array(
            fopra=>"SELECT, INSERT",
            control=>"INSERT, UPDATE",
          ),
        ),
      ),
      check=>array(
        viewid=>"vorlesungsanmeldung_check",
        leittabelle=>"student",
        query=>array(
          array(
            SELECT=>"s.studentnummer, s.matrikelnummer, s.universitaetsnummer",
            FROM=>"student s"
          ),
        ),
        rechte=>"SELECT",
      ),
    );
    modul_extmod($modul_param);
```

4.4.5 Der Eintrag 'new_tupel' im Detail

Der Schlüssel 'new_tupel' im Parameter-Array für die Funktion 'modul_extmod' nimmt die Details für die Erzeugung einer temporären Tabelle auf. Der Name der erzeugten temporären Tabelle wird in der Relation 'workflow_extmod_param' unter dem Namen 'table_new' abgespeichert. Die temporäre Tabelle kann im Zusammenspiel mit 'insert' und 'check' verwendet werden. Das Beispiel aus Abschnitt 4.4.4 könnte um eine temporäre Tabelle erweitert werden, um Daten aufzunehmen, die durch die Überprüfung mit dem Check-View durchgefallen sind. Eine temporäre Tabelle kann aber auch ohne Insert- und Check-Views genutzt werden, wenn man grundsätzlich keine Daten von außen in die regulären Tabellen der DB gelangen lassen will. Die Daten der temporären Tabelle müssen in beiden Fällen "manuell", am besten nach vorheriger Überprüfung in die regulären Tabellen übernommen werden.

Da die temporäre Tabelle Daten enthält die später in die regulären Tabellen übernommen werden sollen, müssen sich die Attribute der temporären Tabelle auf Attribute der regulären Tabellen beziehen. Unter dem Schlüssel 'attribs' von 'new_tupel' ist deshalb ein Array abgelegt. Dieses Array enthält als Schlüssel die Namen der regulären Tabellen und als Werte die Attribute, die für die temporäre Tabelle genutzt werden sollen. Die Attribute sind durch Komma getrennt in einem String abgelegt. Tabelle 16 beschreibt die Bestandteile von 'new_tupel'.

Bemerkungen:

'tmptableid': Die Generierung der Namen der temporären Tabellen durch die Workflowanwendung soll sicherstellen, dass die Namen in der DB eindeutig sind. Die Temptable-IDs müssen hingegen nur für einen Workflow eindeutig sein. Dadurch kann ein Modul mit einem ExtMod-Submodul in mehreren Workflows verwendet werden, ohne dass besondere Vorkehrungen für die Namen der temporären Tabellen getroffen werden müssen. Die Namen der temporären Tabellen werden in die Relation 'workflow_extmod_param' eingetragen. Die Namen können außerdem mit Hilfe der Funktion 'view_info($tmptableid)' abgefragt werden. Die Funktion gibt ein Array zurück, das unter dem Schlüssel 'name' den Namen der temporären Tabelle enthält.

Schlüssel	Beschreibung
tmptableid	String, mandatory. Mit Hilfe dieser ID werden die temporären Tabellen identifiziert. Die IDs müssen innerhalb eines Workflows eindeutig sein. Die eigentlichen Namen der Tabellen werden von der Anwendung generiert und haben die Form: tmp<aktuelle_zeit>.
attribs	Array, mandatory. Enthält als Schlüssel die Namen der regulären Tabellen und als Werte die Attribute daraus, die für die temporäre Tabelle benutzt werden sollen. Die Attribute durch Komma getrennt als Strings einzutragen
primkeys	Array. Falls notwendig kann hier in Form einer Liste angegeben werden, welche Attribute gleichzeitig Primärschlüssel sein sollen.
rechte	String oder Array. Die hier definierten Rechte werden für die temporäre Tabelle gesetzt. Bei der ersten Variante werden die Rechte für den unter 'user' (siehe Abschnitt 4.3.2) eingetragenen Benutzer gesetzt. Bei der Array-Variante werden die Benutzernamen als Schlüssel und die zugeordneten Rechte als Werte eingetragen. Die Rechte sind in beiden Varianten durch Komma getrennt.

Tabelle 16: Bestandteile von 'new_tupel'

'attribs': Beim Erzeugen einer neuen Tabelle muss neben dem Namen auch der Typ für jedes Attribut angegeben werden. Die Namen der Attribute der temporären Tabelle ergeben sich aus den Namen der Attribute der regulären Tabelle, auf die sich die temporäre Tabelle bezieht. Auch die Typen werden entsprechend den Typen der regulären Tabellen gesetzt.

Im folgenden Beispiel wird eine temporäre Tabelle erzeugt, die die Daten der Studenten aufnimmt, die sich zu einer Vorlesung anmelden. Als Primärschlüssel werden die Attribute 'matrikelnummer' und 'universitaetsnummer' benutzt. Der Verzicht auf den Primärschlüssel kann durchaus sinnvoll sein, um zu verhindern, dass bösartige Studenten mit falschen Einträgen die Eingabe von anderen Studenten verhindern. Dadurch kann es in der temporären Tabelle mehrere Tupel mit der selben Matrikelnummer-Uni-Kombination geben. Der Betreuer der Vorlesung muss dann den plausibelsten auswählen.

```
$modul_param = array(
  id=>"steuerung_extmod",
  extmod_id=>"extmod1",
  aufruf_id=>"test",
  new_tupel=>array(
    tmptableid=>"vorlesungsanmeldung_new",
    attribs=>array(
      student=>array("matrikelnummer", "universitaetsnummer", "name",
        "vorname", "hauptfach", "nebenfach", "aufbaustudent",
        "cipkennung", "telefonnummer", "postleitzahl", "ort", "strasse"),
      student_besucht_uebung=>array("besuchtuebung",
        "gruppennummerpraeferenz1", "gruppennummerpraeferenz2"),
      mailadressenstudent=>array("mailadresse"),
    ),
    primkeys=>array("matrikelnummer", "universitaetsnummer"),
```

```
        rechte=>"SELECT, INSERT",
      ),
    );
    modul_extmod($modul_param);
```

4.5 Submodule für den Aufruf externer Programme

4.5.1 Einleitung

Zur Arbeitserleichterung gibt es am Lehrstuhl bereits verschiedene Programme und Skripte, die die Mitarbeiter bei wiederkehrenden Tätigkeiten entlasten. Es ist wünschenswert, dass diese Programme aus der Workflowanwendung heraus mit den notwendigen Parametern aufgerufen werden können. Dies leistet die Funktion 'modul_extprog($modul_param)' in 'lib-file.inc.php'. Der Programmaufruf zusammen mit der Definition der möglichen Parameter kann in Form eines Submoduls in ein Modul-Skript eingebaut werden. Die Funktion stützt sich dabei unter anderem auf die Library 'lib-dbtable.inc.php'.

Mit Hilfe der Funktionen in 'lib-dbtable.inc.php' werden die zwei Bestandteile der HTML-Ausgabe eines ExtProg-Submoduls erzeugt:

- Es kann eine Tabelle ausgegeben werden, die Daten aus der Datenbank darstellt. Dies geschieht mit Hilfe der Funktion 'tabelle_mit_aktionen' in 'lib-dbtable'. Die Daten sind das Ergebnis eines Query, der an die Funktion 'modul_extprog' übergeben werden kann. Der Benutzer kann in der Tabelle blättern und einzelne Datentupel durch Ankreuzen auswählen. Die Werte der ausgewählten Datentupel können dem externen Programm als Parameter übergeben werden, wobei das externe Programm für jeden einzelnen Datentupel gestartet wird.

- Es wird ein Formular ausgegeben, in das Werte eingegeben werden können, die dem Programm als Parameter übergeben werden. Dies wird mit der Funktion 'aktion_neues_tupel' aus 'lib-dbtable' realisiert. In Kombination mit der Tabelle wird jedes ausgewählte Datentupel aus der Tabelle mit den Werten des Formulars ergänzt.

Damit die Workflow-Anwendung weiß, wie die Daten in den Programmaufruf eingebaut werden müssen, werden Platzhalter verwendet. Diese referenzieren die Felder der Tabelle und/oder des Formulars.

Vor den Aufruf des Programms können zusätzlich Datei-Vorlagen abgearbeitet werden, die die selben Platzhalter enthalten können. Aus der Vorlage wird eine Datei erzeugt, in der die Platzhalter durch die aktuellen Daten ersetzt worden sind. Die erzeugte Datei kann dann ebenfalls im Programmaufruf referenziert werden. Da die Daten aus der Datenbank manchmal in einem für den Programmaufruf ungeeigneten Format vorliegen, können sie mit Hilfe von Datenkonvertierungsfunktionen und Lookup-Definitionen in 'form_param', wie sie schon aus den DBTable-Submodulen bekannt sind, bearbeitet werden. Zusätzlich können die geraden aktuellen Daten vor und nach dem Programmaufruf einer beliebigen Funktion übergeben werden.

4.5.2 Zusammensetzung des Parameter-Array $modul_param

Da zum Teil Funktionen aus 'lib-dbtable' genutzt werden, sind viele der Parameter bereits aus Abschnitt 4.3 bekannt. Einige dieser Parameter werden für das Formular, einige für die Tabelle und andere für beides verwendet. Tabelle 17 zeigt diese Parameter im Überblick und gibt wieder, welche Parameter wo verwendet werden. Die detaillierten Beschreibungen zu diesen Parametern finden sich in Abschnitt 4.3. Abweichungen werden im folgenden beschrieben:

- query, leittabelle: Eine Anfrage und eine Leittabelle müssen nur angegeben werden, wenn Daten aus der Datenbank für den Programmaufruf benötigt werden. Beide Parameter müssen in diesem Fall definiert werden.

Parameter	Verwendung im Formular	Verwendung in der Tabelle
leittabelle	nein	ja
values	ja	nein
ziellink	ja	nein
query	nein	ja
cols	ja	nein
aendern_cols	ja	nein
table_param	ja	ja
form_param	ja	ja
appearance	ja	ja
dataconversion	nein	ja

Tabelle 17: Aus DBTable-Submodulen bekannte Parameter und ihre Verwendung

- table_param, form_param: Diese beiden Parameterblöcke wirken auf die Tabelle und das Formular. Deshalb müssen die Namen der Felder aus dem Formular und der Tabelle insgesamt eindeutig sein. Definiert man für ein Attribut der Tabelle in 'form_param' einen Lookup-Query, so macht dies nur im Zusammenhang mit dem Typ 'constant' Sinn.
- appearance: Dieser Parameter hat einige Werte fest vorgegeben, die auch nicht verändert werden können:

```
$appearance = array(
  no_buttons => 1,
  abstaende => array(
    bottom => 0,
  ),
  reihenfolge => array(
    ``tabelle'', ``blaettern_a''
  ),
);
```

Die weiteren Einträge in '$modul_param' sind in Tabelle 18 aufgelistet und beschrieben.

4.5.3 Der Eintrag 'datei_list' im Detail

Im Array 'datei_list' kann an die Funktion 'modul_extprog' eine Liste von Datei-Templates übergeben werden. Die in den Templates enthalten Platzhalter werden durch die aktuellen Daten ersetzt. Dabei wird eine Datei erzeugt, deren Namen folgende Gestalt hat:

```
<Name des Templates>_<PID des HTTP-Servers>_<aktuelle Zeit>
```

Die Dateien werden im selben Verzeichnis abgelegt, in dem sich auch die Vorlage befindet. Deshalb ist darauf zu achten, dass der HTTP-Server auf diesem Verzeichnis auch Schreibrechte besitzt. Die aus den Templates entstandenen Dateien können im Programmaufruf durch die Platzhalter $(file0), $(file1) usw. referenziert werden (unter der Voraussetzung, dass das Format für den Platzhalter $(%s) ist; siehe 4.5.4). Die Nummern beziehen sich auf die Positionen der Datei-Vorlagen innerhalb des Arrays 'datei_list'. Die erzeugten Dateien werden nicht automatisch gelöscht, dies muss entweder später von Hand erledigt werden oder in den Programmaufruf einbezogen werden. Die Datei-Liste wird auch bearbeitet, wenn kein Programmaufruf angegeben ist.

Als Beispiel für die Verwendung von 'datei_list' könnte man sich vorstellen, dass LaTeX-Templates existieren. Daraus werden Dateien erzeugt, in denen die Platzhalter durch Daten aus der Datenbank ersetzt wurden. Diese können dann durch einen Programmaufruf ausgedruckt oder zur späteren Verwendung nur gespeichert werden.

Parameter	Typ	Default	Beschreibung
id	String		Wie jedes DBTable-Submodul muss auch jedes ExtProg-Submodul eine eindeutige ID innerhalb des Workflow-Schritts haben.
datei_list	Array		Eine Liste mit den Datei-Vorlagen, die for dem Programmaufruf abgearbeitet werden sollen. Die Templates können Platzhalter enthalten.
prog_aufruf	String		Programmaufruf. Innerhalb des Aufrufs können Platzhalter verwendet werden.
var_format	String	$(%s)	Aussehen der Platzhalter.
system_call	String	system	In PHP stehen für Aufrufe externer Programme die Funktionen 'system', 'exec' und 'passthru' zur Verfügung.
prepare_function	String		Vor und nach dem Aufruf des externen Programms kann eine beliebige Funktion aufgerufen werden.
usertext	Array		Die Struktur des Arrays ist identisch zu 'usertext' in einem DBTable-Submodul. Hier bezieht sich 'usertext' allerdings auf das ganze Submodul und nicht auf die einzelnen Bestandteile.

Tabelle 18: weitere mögliche Einträge in '$modul_param'

Funktion	Ausgabe
system	Das Ergebnis des Programmaufrufs wird zeilenweise zurückgegeben. Nach jeder Zeile wird versucht den Ausgabe-Puffer des Web-Servers zu leeren.
exec	keine Ausgabe
passthru	Das Ergebnis des Programmaufrufs wird "raw" an den Browser zurückgegeben. Geeignet für binäre Daten (gifs etc.).

Tabelle 19: PHP-Funktionen zum Aufruf externer Programme

4.5.4 Der Eintrag 'var_format' im Detail

Da es in den Dateivorlagen und im Programmaufruf unter Umständen zu Konflikten mit dem Standard-Format für die Platzhalter ($(%s)) kommen kann, hat der Modul-Programmierer die Möglichkeit ein anderes Format vorzugeben. In diesem Format muss auf jeden Fall ein '%s' enthalten sein, da dieser String mit Hilfe der PHP-Funktion 'sprintf' durch den Feldnamen ersetzt wird. Angenommen es existiert ein Feld mit dem Namen 'xyz', so ist unter Verwendung des Standard-Formats der Platzhalter '$(xyz)' zu verwenden.

4.5.5 Der Eintrag 'system_call' im Detail

Der Programmaufruf wird standardmäßig mit der PHP-Funktion 'system' erledigt, die der aus C bekannten Funktion entspricht. PHP bietet daneben auch noch die Funktionen 'exec' und 'passthru'. Die Funktionen unterscheiden sich vor allem in der Art der Ausgabe, wie in Tabelle 19 dargestellt.

4.5.6 Der Eintrag 'prepare_function' im Detail

Unmittelbar vor und nach dem Aufruf des externen Programms können die Daten einer in 'prepare_function' angegebenen Funktion übergeben werden. Beidesmal wird die selbe Funktion verwendet. Eine solche Funktion kann beispielsweise direkt in einem Modul-Skript stehen. Sie wird von 'modul_extprog' aufgerufen, wobei ihr zwei Parameter übergeben werden:

Eine Referenz auf das Array, in dem die aktuellen Daten liegen. Dieses Array ist mit den Feldnamen indiziert.

Ein boolscher Wert, über den die Funktion erkennen kann, ob der Aufruf des externen Programms noch folgt oder schon beendet ist. Im ersten Fall ist der Parameter nicht definiert, im zweiten Fall auf 'false' gesetzt.

Mit Hilfe einer solchen Funktion könnten beispielsweise die Daten vor dem Aufruf des externen Programms und zusätzlich zu den Möglichkeiten von 'dataconversion' und 'form_param' bearbeitet werden. Eine weitere Anwendung wäre die Speicherung der aktuellen Daten in einer Datei nach erfolgtem Programmaufruf. Die im folgenden Beispiel dargestellte Funktion erzeugt eine Datei mit einer Liste, der zu einer Klausur angemeldeten Studenten. Im Datenarray (als Referenz übergeben) findet die Funktion die Nummern der Vorlesung ('vn_nr') und der Klausur ('kl_nr'). Außerdem enthält das Array Platz für den Namen der erzeugten Datei ('file0'). Wenn die Funktion vor dem externen Programmaufruf gestartet wird, erzeugt sie die Datei. Nach dem Aufruf ($vorher == 0) wird die Datei wieder gelöscht.

```
function liste_erzeugen (&$daten, $vorher=1) {
  if (!$vorher) {
    unlink($daten[file0]);
    return;
  }
  $query = "SELECT s.name, s.vorname,
      s.matrikelnummer, ssk.platznummer
    FROM student s, student_schreibt_klausur ssk
    WHERE ssk.klausurnummer='".$daten[kl_nr]."'
      AND ssk.vorlesungnummer='".$daten[vn_nr]."'
      AND ssk.studentnummer=s.studentnummer";
  $res = db_query($query);
  if (!$res){
    debug("Fehler beim Ausführen des Query");
    return;
  }
  $tmpfname = tempnam("/tmp", "");
  if (!$tmpfname){
    debug("Fehler beim Anlegen der temp. Datei ");
    return;
  }
  $fd = fopen($tmpfname, "w");
  if (!$fd){
    debug("Fehler beim Öffnen der temp. Datei ");
    return;
  }
  while ($zeile = db_fetchnext_array($res)){
    $string = "$zeile[platznummer] $zeile[vorname] $zeile[name]\n";
    fputs($fd, $string);
  }
  fclose($fd);
  $daten[file0] = $tmpfname;
}
```

Der Parameter-Block für den Aufruf von 'modul_extprog' könnte dann wie folgt aussehen:

```
$modul_param = array(
  id=>"liste",
  values=>array(
```

```
    vn_nr=>10,
    kl_nr=>1,
  ),
  cols=>array("vn_nr", "kl_nr"),
  aendern_cols=>array("kl_nr"),
  table_param=>array(
    headcell_format=>array(
      kl_nr=>"Klausurnr.",
      vn_nr=>"Vorlesungsnr.",
    ),
  ),
  prepare_function=>"liste_erzeugen",
  prog_aufruf=>"cat $(file0)",
);
modul_extprog($modul_param);
```

4.6 HTML-Schnittstelle

Die Grundlagen für das Erzeugen von HTML-Code im System werden in der Bibliothek ´lib-html.inc.php´ gelegt. Im folgenden wird diese Library kurz ´lib-html´ genannt.

Die in ´lib-html´ enthaltenen Funktionen produzieren alle HTML-Code, d.h. sie liefern in aller Regel als Rückgabe einen String, der den erstellten HTML-Code enthält. Will der Benutzer der jeweiligen Funktion das Ergebnis dann tatsächlich auf der Standardausgabe darstellen, so muss er dies selbst noch tun - z.B. mit dem php-internen Befehl ´echo´ . In vielen Fällen wird er aber den HTML-Code erst durch mehrere Funktionsaufrufe in lib-html erst in einem oder mehreren Strings zusammenstellen, um diese Strings später weiterzuverarbeiten. Beispielsweise könnte man in einem String eine Schablone für einen bestimmten HTML-Code (z.B. für ein Formular) erstellen, in die erst später tatsächlich Werte aus der Datenbank eingesetzt werden. Daher liefern i.a. die Funktionen aus ´lib-html´ alle nur Strings zurück und machen selbst keine Ausgabe aus stdout.

Die Bibliothek ist so angelegt, das sie den gesamten für das System notwendigen HTML-Code vollständig selbst produziert, d.h. die ´libhtml´-Funktionen erzeugen alle HTML-Tags und deren korrekte Verschachtelung. Dies gilt nicht nur für Tabellen und Formulare, sondern ´libhtml´ kann auch zum Erstellen von weniger komplizierten HTML-Konstrukten, wie Links, Images, Überschriften, etc. verwendet werden. Der Benutzer muss selbst keinen HTML-Code erstellen, d.h. v.a. keine HTML-Tags selbst benutzen, geschweige denn auf derren Verschachtelung achten. Im Workflowsystem wird i.a. stets eingehalten, d.h. HTML-Code wird (fast) nur ´libhtml´-Funktionen produziert, und HTML-Tags nie außerhalb, direkt verwendet. Es steht jedoch jedem Benutzer, d.h. z.B. jemandem, der ein neues Workflowmodul erstellt, frei, in seinem Programm-Code selbst direkt HTML-Code zu verwenden. Dies gilt natürlich vor allem falls HTML-Konstrukte verwendet werden sollen, die ´libhtml´ nicht kennt. Gerade bei komplexeren Formularen und Tabellen sind die Funktionen aus ´libhtml´ aber auf alle Fälle zu empfehlen.

Alle von ´libhtml´-Funktionen erstellten HTML-Tags werden - konform zum XHTML-Standart - klein geschrieben.

Unter HTML-Encoding wird im folgenden die für HTML übliche Kodierung von Sonderzeichen (z.B. “<” als “>”) und Umlauten, etc. verstanden. Mit URL-Encoding ist die für Parameterübergaben per URL übliche Kodierung von Zeichen gemeint (z.B. “ “ als “%20 “).

Die Funktionen von ´libhtml´ lassen sich in die Bereiche Tabellenerzeugung, Formularerzeugung, Erzeugung von sonstigen HTML-Konstrukten, Erstellen von Parameter-Array-übergaben per Link oder Form, Ersetzen von Variablenwerten und Debugmeldungs-Generierung einteilen. Das Ersetzen von Variablenwerten findet vor allem Anwendung bei Tabellen und Formularen.

Im folgenden wird auf die einzelnen Bereiche genauer eingegangen.

Die Namen der ´libhtml´-Funktionen beginnen immer mit ´html_´.

4.6.1 Funktionen zum Erstellen von Tabellen:

Die Unterstützung von HTML-Tabellen wurde vor allem für das Anzeigen von SQL-Anfrage-Ergebnissen konzipiert. Man könnte z.B. mit dem einzigen Funktionsaufruf ´html_table($data)´ das Result einer SQL-Abfrage als Tabelle anzeigen, wenn $data eine Liste von SQL-Ergebnis-Datentupeln (Ergebnis-Zeilen), also eine Liste von Rückgaben der Funktion db_query, ist. Dabei werden in einer Kopfzeile die Namen der Attribute angezeigt. und jeweils in jeder folgenden (Daten)-Zeile je ein Datentupel. Will man das Aussehen der Tabelle dann weiter beeinflussen, sind weitere Parameter nötig.

Um mit ´libhtml´ HTML-Code für eine Tabelle zu erstellen, kann man entweder einen einzigen Funktionsaufruf an die Funktion ´html_table´ machen oder man ruft selbst die Funktionen zum Erstellen der Teile einer Tabelle ´html_table_head´, ´html_table_lines´ bzw. ´html_table_line´ und ´html_table_end´ auf. Die Funktion ´html_table´ ist selbst ein Wrapper, der die anderen genannten Funktionen aufruft. Vor allem bei großen Datenmengen wird man möglicherweise die zweite Variante bevorzugen.

html_table_head($cols = array(), $param = array()): Mit dieser Funktion wird der HTML-Quelltext für den Kopf einer HTML-Tabelle, d.h. den Beginn der Tabelle (<table...>-Tag) und eine Kopfzeile, erzeugt. Die Parameter von ´html_table_head´ sind:

$cols: Liste von Spaltennamen der Tabelle. Diese Bezeichner werden an anderer Stelle als Referenz benutzt. Die Reihenfolge in dieser Liste bestimmt die Reihenfolge der Spalten in der HTML-Tabelle (von links nach rechts).

$param: assoziatives Array, das weitere Parameter für die Tabelle enthalten kann:

- border: boolsches Flag, falls false, so erhält die Tabelle keinen Rand (border-Attribut im Table-Tag)
- bgcolor: Hintergrundfarbe der Tabelle (bgcolor-Attribut im Table-Tag)
- bordercolor: Farbe des Tabellenrandes (bordercolor-Attribut im Table-Tag)
- alignment: Ausrichtung der Tabelle (align-Attribut im Table-Tag): right, left, top, bottom oder center
- width: Weite der Tabelle (width-Attribut im Table-Tag), z.b. 50%
- cellpadding: Randabstand vom Zelleninhalt zum Zellenrand (cellpadding-Attributes im Table-Tag)
- cellspacing: Dicke der Gitternetzlinien (cellspacing-Attributes im Table-Tag)
- rules: Wert des rules-Attributes im table-Tag ?, z.B none?
- caption_top: mögliche obere Beschriftung der Tabelle (<caption align=top>...</caption>)
- caption_bottom: mögliche untere Beschriftung der Tabelle (<caption align=bottom>...</caption>)
- caption_left: mögliche linke Beschriftung der Tabelle (<caption align=left>...</caption>)
- caption_right: mögliche rechte Beschriftung der Tabelle (<caption align=right>...</caption>)
- no_empty_headline: boolsches Flag, falls 'true', wird eine Kopfzeile nicht erzeugt, falls $cols leer ist, d.h. eine leere Kopfzeile entstehen würde. Findet z.B. Anwendung, wenn ´html_table_head´ direkt aufgerufen wird.
- head_bgcolor: Hintergrundfarbe der Kopfzeile (bgcolor-Attribut im tr-Tag)
- head_fgcolor: Standart-Vordergrundfarbe der Kopfzeile (<font color=..>)

head_valign: valign-Ausrichtung der Kopfzeile (valign-Attribut im tr-Tag)

headcell_bgcolor: assoziatives Array, das für eine Spalte anhand ihres Namens aus $cols eine eigene Hintergrundfarbe festlegt (bgcolor-Attribut im th-Tag)

headcell_fgcolor: assoziatives Array, das für eine Spalte anhand ihres Namens aus $cols eine eigene Vordergrundfarbe festlegt (<font color=...>)

headcell_format: assoziatives Array, das für eine Spalte anhand ihres Namens aus $cols ein Ausgabe-Text festlegt, der Standart hierfür ist jeweils der Name der Spalte selbst

headcell_param: assoziatives Array, das für eine Spalte anhand ihres Namens aus $cols weitere Parameter für die Spalte festlegt, d.h. Wert zu einem Key (Spaltenname) ist selbst assoziatives Array mit folgenden möglichen Parametern:

width: Weite der Kopfspalte (width-Attribut im th-Tag)

p_align: Ausrichtung des Inhalts der Kopfspalte mit <p align=..>-Tag

bold: Inhalt der Kopfspalte wird fett gedruckt (..)

Als Rückgabe liefert die Funktion einen String, der den HTML-Quelltext für den Tabellen-Kopf und die Kopfzeile enthält.

html_table_line($entry, $cols0, $param = array ()): Mit ´html_table_line´ wird HTML-Code für eine (Daten)-Zeile einer Tabelle erzeugt. Dabei können Daten aus einem gegebenen Datentupel (z.b. aus der Datenbank, wie es von db_query zurückgeliefert wird) in gegebene Schablonen eingesetzt werden (siehe Parameter $param[datacell_format] unten). Parameter sind:

$entry: Datentupel, d.h. ein assoziatives Array bestehend aus beliebigen Key/Value-Paaren ist, z.b. wie es von db_query als Repräsentation einer Zeile eines SQL-Ergebnisses zurückgeliefert wird

$cols0: Liste Spaltennamen der Tabelle. Diese Bezeichner werden an anderen Stellen als Referenz benutzt. Die Reihenfolge in dieser Liste bestimmt die Reihenfolge der Spalten in der HTML-Tabelle, Eine Spalte mit Namen '*' wird aber zu Beginn durch die Liste aller (nichtnummerischen) Keys des Datentupels $entry ersetzt.

$param: assoziatives Array, das weitere Parameter enthalten kann:

data_bgcolor: Standart-Hintergrundfarbe der Tabellenzeile (bgcolor-Attribut im tr-Tag)

data_bgcolor2: alternative Zeilen-Hintergrundfarbe; falls festgelegt, so wechselt die Hintergrundfarbe pro Zeile (d.h. pro Aufruf der Funktion db_table_line) zwischen dem in data_bgcolor2 und dem in data_bgcolor festgelegten Wert data_fgcolor: Standart-Vordergrundfarbe der Tabellenzeile (...)

data_fgcolor2: alternative Zeilen-Vordergrundfarbe; falls festgelegt, so wechselt die Vordergrundfarbe pro Zeile (d.h. pro Aufruf der Funktion db_table_line) zwischen dem in data_fgcolor2 und dem in data_fgcolor festgelegten Wert

data_valign: valign-Ausrichtung der Tabellen-Zeile (valign-Attribut im tr-Tag)

data_align: align-Ausrichtung der Tabellen-Zeile (align-Attribut im tr-Tag)

datacell_bgcolor: assoziatives Array, das für eine Spalte anhand ihres Namens aus $cols eine eigene Hintergrundfarbe festlegt (bgcolor-Attribut im td-Tag)

datacell_fgcolor: assoziatives Array, das für eine Spalte anhand ihres Namens aus $cols eine eigene Vordergrundfarbe festlegt (<font color=...>)

datacell_format: Gibt ein Format für den Inhalt einer Tabellenspalte an.
Kann im Prinzip beliebiger String sein, in dem vor Benutzung eine Reihe von Ersetzungen möglich ist. Vor allem sind Werte aus dem Datensatz $entry einsetzbar. Die Ersetzung wird durch die Funktion ´html_substitute_data´ vorgenommen, wobei hier alle von dieser Funktion unterstützten Ersetzungen erlaubt sind (siehe ´html_substitute_data´). Z.B. wird "$(feldname)" durch den bloßen String-Wert des Wertes $entry[feldname] aus dem Datensatz ersetzt, "$[feldname]" wird durch den htmlencodeten Wert von entry[feldname] und "$<feldname>" durch den urlencodeten Wert von $entry[feldname] ersetzt. Weitere Ersetzungen siehe ´html_substitute_data´.
Ist für eine Spalte hier kein Format angegeben, so wird der Wert von datacell_format_default (siehe unten) statt dessen benutzt, fehlt auch dieser, so wird der Wert aus dem Datensatz $entry[$spaltenname] (htmlencoded) benutzt, wenn $spaltenname der Name der aktuellen Spalte aus $cols0 ist.

datacell_param: assoziatives Array, das für eine Spalte anhand ihres Namens aus $cols weitere Parameter für die Spalte festlegt, d.h. Wert zu einem Key (Spaltenname) ist selbst assoziatives Array mit folgenden möglichen Parametern:

- width: Weite der Spalte (width-Attribut im td-Tag)
- align: Ausrichtung der Spalte mit align-Attribut des td-Tags
- valign: Ausrichtung der Spalte mit valign-Attribut des td-Tags
- p_align: Ausrichtung des Inhalts der Spalte mit <p align=..>-Tag
- bold: Inhalt der Kopfspalte wird fett gedruckt (..)
- colspan: Verbindung dieser Spalte mit einer oder mehreren der folgenden, bzw. colspan-Attribut des td-Tags und entsprechend viele folgende Spalten aus $cols werden übersprungen.
  Vor allem dann sinnvoll, falls jede Spalte durch einzelnen Aufruf von ´html_table_line´ durch Benutzer mit jeweils eigenem $param aufgebaut wird
- datacell_format_default: Default-Wert für die Werte des Arrays datacell_format, falls nicht spezifiziert wird als Default für den Inhalt einer Spalte mit Namen $key jeweils der Wert aus dem Datentupel $entry[$key] genommen
- dataconversion: assoziatives Array, das für jede Komponente eines Datensatzes anhand ihres Namens aus $cols0 eine Datenkonvertierungsfunktion definieren kann.
  Eine solche Funktion wird zu Beginn auf die jeweiligen Komponenten des Datensatzes $entry angewendet, bevor dieser benutzt wird. Sie ersetzt den jeweiligen Wert, durch die Rückgabe der Konvertierungsfunktion, die mit demursprünglichen Wert als erstem Parameter aufgerufen wird.
  Vor allem gedacht, um bei SQL-Ergebnisse Daten für die Ausgabe zu konvertieren, z.B. SQL-DATE oder SQL-BOOL-Werte.

Als Rückgabe liefert diese Funktion einen String, der den HTML-Quelltext für die Tabellen-Zeile enthält.

html_table_lines($datas, $cols, $param = array ()): Will man für mehrere Datensätze statt vielen einzelnen Aufrufen zur Funktion ´html_table_line´ mit jeweils gleichen $cols und $param einen einzigen Aufruf machen, so ist dies mit der Wrapper-Funktion ´html_table_lines´ möglich. Diese ruft für jedes Datentupel aus einer gegebenen Liste die Funktion ´html_table_line´ auf. Die Rückgabewerte der einzelnen Aufrufe, d.h. die HTML-Quelltexte der einzelnen (Daten).Zeilen,

werden zu einem String zusammengesetzt und zurückgeliefert.
Parameter sind (siehe auch ´html_table_line´):

$datas: Liste von Datentupeln, wobei ein Datentupel Datentupel ein assoziatives Array, bestehend aus beliebigen Key/Value-Paaren ist, z.b. wie es von db_query als Repräsentation einer Zeile eines SQL-Ergebnisses zurückgeliefert wird, d.h. die Elemente von $datas werden einzeln als Parameter $entry an ´html_table_line´ übergeben.

$cols: Liste Spaltennamen der Tabelle. Diese Bezeichner werden an anderen Stellen als Referenz benutzt. Die Reihenfolge in dieser Liste bestimmt die Reihenfolge der Spalten in der HTML-Tabelle

$param: assoziatives Array, das weitere Parameter enthalten kann: siehe ´html_table_line´

Die Funktion liefert als Rückgabe einen String, der den HTML-Quelltext für die Tabellen-Zeilen enthält

***html_table_line_columns ($entry, $cols, $param):** Hilfsfunktion für ´html_table_line´, die die Inhalte der einzelnen Spalten erzeugt, d.h. ggf. Datenkonvertierungsfunktionen auf den Datensatz anwendet und dann jede aus ihrem Format und dem Datensatz aufbaut. Sie hat folgende Parameter:

$entry: Datensatz der Spalte, beliebiges assoziatives Array

$cols: Liste der Spaltennamen, siehe ´html_table_line´

$param: Parameter, siehe ´html_table_line´; Tatsächlich verarbeitet werden hier dataconversion, datacell_format, datacell_format_default.

Rückgabe von ´html_table_line_columns´ ist assoziatives Array der Spalteninhalte (Keys sind die Werte aus $cols).

html_table_end(): Diese Funktion gibt String einen mit HTML-Code für ein Tabellenende (</table>.-Tag) zurück. Sie wird verwendet, um eine mit ´html_head´, ´html_table_line´ bzw. ´html_table_lines´ begonnene Tabelle abzuschließen.

html_table($datas, $cols0, $param = array ()): Statt eine Tabelle durch Einzelaufrufe von ´html_table_head´, ´html_table_line´ bzw. ´html_table_lines´ und ´html_table_end´ zu erzeugen, kann bei gleichbleibendem $cols und $param auch ein einziger Aufruf der Funktion ´html_table´ verwendet werden. Mit ´html_table´ wird also der HTML-Quelltext für eine ganze HTML-Tabelle erzeugt, d.h. Tabellenkopf (<table>-Tag), Kopfzeile und Datenzeilen sowie Tabellenende (</table>-Tag). Für den Kopf und die Kopfzeile wird ´html_table_head´ aufgerufen. Die (Daten)-Zeilen werden dabei aus einer Liste vorgegebener Datentupel mit der Funktion ´html_table_lines´ generiert. Von´html_table_lines´ wird dabei je eine Zeile aus einem Datentupel generiert, d.h. pro Datentupel wird ´html_table_line´ aufgerufen. Die Rückgabewerte der einzelnen Aufrufe, d.h. die HTML-Quelltexte von Tabellenkopf, Kopfzeile, den einzelnen Daten-Zeilen und vom Tabellenende werden zu einem String zusammengesetzt und zurückgegeben.
´html_table´ ist vor allem für die Ausgabe von SQL-Ergebnissen gedacht.
Parameter sind (siehe auch ´html_table_head´, ´html_table_lines´, ´html_table_line´, ´html_table_end´) :

$datas: Liste von Datentupeln, wobei ein Datentupel Datentupel ein assoziatives Array, bestehend aus beliebigen Key/Value-Paaren ist, z.b. wie es von db_query als Repräsentation einer Zeile eines SQL-Ergebnisses zurückgeliefert wird, d.h. die Elemente von $datas werden einzeln als Parameter $entry an ´html_table_line´ übergeben.

$cols0: Liste Spaltennamen der Tabelle. Diese Bezeichner werden an anderer Stelle als Referenz benutzt. Die Reihenfolge in dieser Liste bestimmt die Reihenfolge der Spalten in der HTML-Tabelle.
Eine Spalte mit Namen '*' wird aber ganz zu Beginn durch die Liste aller (nichtnummerischen) Keys des ersten Datentupels $datas[0] ersetzt. Ist die $cols leer, so wird als Standart '*' und damit dann die Liste aller (nichtnummerischen) Keys von $data[0] gewählt .

$param: assoziatives Array, das Parameter für die Funktionen ´html_table_head´ (Tabellenkopf und Kopfzeile) und ´html_table_line´ (Datenzeilen) enthalten kann. Siehe ´html_table_head´ bzw. ´html_table_line´.

Rückgabe von ´html_table´ ist ein String, der den HTML-Quelltext für die Tabelle enthält.

Oft betrachtet man nicht eine ganze Liste von Datensätzen, sondern nur einen einzigen. Um einen Datensatz alleine in einer Tabelle darzustellen, empfiehlt es sich nicht, die Tabelle aus Kopfzeile und einer Datenzeile für den einzigen Datensatz aufzubauen. Besser ist eine mehrspaltige Darstellung, bei der Beschriftung, die sonst in der Kopfzeile stehen würde, und Datenwert in aufeinanderfolgende Spalten der Spalten zu setzen. Dies erledigt für den Benutzer die folgende Funktion:

html_mtable($data, $cols0 = array(), $param = array ()): ´html_mtable´ kann statt ´html_table´ verwendet werden, um einen oder mehrere Datensätze, die in einer einzigen Tabelle jeweils in einer Zeile einer dargestellt werden würde, in jeweils einer eigenen Tabelle mehrspaltig anzuzeigen:
Hierbei fehlt die Kopfzeile, statt dessen gibt es in der entstehenden HTML-Tabelle zwei Arten von Spalten, nämlich Beschriftungsspalten, die bei normalen Tabellen in der Kopfzeile stehen würden, und Datenspalten dazu. Dabei wird vor jeder definierten Daten-Spalte aus $cols0 eine Spalte mit der Beschriftung der Spalte, die sonst in der Kopfzeile stehen würde ausgegeben. Das Verteilen der gegebenen Daten-Spalten auf die tatsächlichen Spalten der HTML-Tabelle geschieht automatisch. Die Parameter sind im wesentlichen die gleichen wie bei ´html_table_line´ mit kleinen Erweiterungen (siehe ´html_table´):

$data: Liste von Datentupeln wie bei ´html_table´

$cols0: Liste der vorgegebenen Spalten wie bei ´html_table´

$param: Array von Parametern, mit den gleichen Parametern wie bei ´html_table´ (d.h. ´html_table_head´ und ´html_table_line´), sowie folgende Parameter:

mtable_format: "horizontal" (bzw. "h") oder "vertical" (bzw. "v"); Gibt an, wie die reale HTML-Tabelle mit den gegebenen Daten-Spalten gefüllt wird: horizontal bzw. vertikal, Default: horizontal;

mtable_columns_per_line: Gibt die Anzahl der tatsächlich in der HTML-Tabelle verwendeten HTML-Spalten (geteilt durch 2) an, d.h. wie viele definierte Daten-Spalten aus $cols0 in einer tatsächlichen HTML-Zeile zusammengefasst werden, Default: 1, d.h. pro gegebener Spalte aus $cols eine Zeile, d.h. alle gegebenen Daten-Spalten untereinander.

mtable_table_heading: beliebiger HTML-Text, der zwischen die einzelnen HTML-Tabellen der Datensätze gesetzt wird. Standart: HTML-Zeilenumbruch (
).

Rückgabe von ´html_mtable´ ist ein String, der den HTML-Code für die mehrzeiligen Tabellen (eine pro Datensatz) enthält.

4.6.2 Funktionen die Erzeugung von Formularen

Die Funktionen zur Erzeugung von Formularen ähneln den Funktionen zur Tabellenerzeugung. Es gibt wieder die Möglichkeit die Teile eines Formulares einzeln durch entsprechende Funktionsaufrufe zu erzeugen oder dies durch den Aufruf eines Wrappers zu erledigen. Auch Formulare erlauben wie Tabellen das Einsetzen von Variablenwerten, um in vorgegebenen Formaten Werte aus einem gegebenen Datensatz einzusetzen.

html_form_head($param): Diese Funktion erzeugt HTML-Code für den Beginn eines Formulares. Sie hat nur einen Parameter:

$param: Array von möglichen Parametern:

- action: action-Attribut des form-Tags (Skriptname, ...)
- method: method-Attribut des form-Tags: "post" oder "get"
- enctype: Encoding, d.h. enctype-Attribut des form-Tags
- target: Zielfenster für Server-Antwort bei Frames, d.h. target-Attribut des form-Tags

Rückgabe ist ein mit HTML-Code für den Formularbeginn.

html_form_element($e, $data = array ()): HTML-Code für einzelne Formular-Elemente erzeugt die Funktion ´html_form_element´. Die Werte des gegebenen Datentupels $data sind dabei in der Ausgabe ersetzbar, d.h. vor Ausgabe der Formular-Quelltextes werden Werte aus $data durch die Funktion ´html_substitute_data´ ggf. hinein-substituiert, wobei alle von ´html_substitute_data´ unterstützten Ersetzungen erlaubt sind, aber nur Variablen ersetzt werden, die auch im Datensatz $data vorhanden sind ($allflag=0 für ´html_substitute_data´) (siehe ´html_substitute_data´). Funktionsparameter sind:

$e: Array von Parametern, die das Formelement beschreiben: Dabei ist der Parameter "type" der wichtigste, denn er bestimmt die Art des Form-Elementes. Im folgeneden wird danach unterschieden:

- type="text" oder type="password": Formelement für Text (<input type=text...> bzw <input type=password...>); weitere Parameter in $e sind hierfür:

  name: name des Formularelements, name-Attribut des input-Tags

  value: voreingestellter Wert, value-Attribut des input-Tags, wird htmlencoded

  size: Länge der Eingabezeile (sichtbar), size-Attribut des input-Tags

  maxlength: maximale Länge der Eingabe, maxlength-Attribut des input-Tags

  readonly: boolsches Flag, falls true, ist Formularelement nur lesbar, d.h. readonly-Attribut des input-Tags wird gesetzt

- type="hidden": Formelement für versteckte Werte (<input type=hidden...>); weitere Parameter in $e sind hierfür:

  name: name des versteckten Wertes, name-Attribut des input-Tags

  value: versteckter Wert, value-Attribut des input-Tags, wird htmlencoded

- type="textarea": Formelement für mehrzeiligen Text (<textarea...>); weitere Parameter in $e sind hierfür:

  name: name der textarea, name-Attribut des textarea-Tags rows: Anzahl Zeilen, rows-Attribut des textarea-Tags

cols: Anzahl Spalten, cols-Attribut des textarea-Tags

wrap: Zeilenumbruchsverhalten, wrap-Attribut des textarea-Tags: z.B. physical, virtual, off

readonly: boolsches Flag, falls true, ist Formularelement nur lesbar, d.h. readonly-Attribut des textarea-Tags wird gesetzt

content: voreingestellter Wert, d.h. Inhalt des textarea-Tags, wird htmlencoded

- type=”select”: Formelement für Auswahlmenü (<select...>; weitere Parameter in $e sind hierfür:

name: name des Auswahlmenüs, name-Attribut des select-Tags

size: Größe des Menüs (sichtbar), d.h. Anzahl Zeilen, size-Attribut des select-Tags

multiple: boolsches Flag, falls true ist dieses Menü ein mehrfaches Auswahlmenü, d.h. multiple-Attribut des select-Tags wird gesetzt

values: Liste der möglichen Auswahlwerte; Dabei ist ein Auswahlelement selbst ein Array mit dem Parameter ”<value>” für den eigentlichen Wert und mit dem Parameter ”<content>” für die Beschriftung.
In jedem dieser Werte werden Datentupel-Werte von $data zu Beginn durch ´html_substitute_data´ ggf. substituiert (z.b. wird $[keyname] in einem Wert durch $data[keyname] (htmlencoded) ersetzt, weitere Ersetzungen siehe ´html_substitute_data´, es werden alle von ´html_substitute_data´ möglichen Substitutionen erlaubt, aber nur Variablen ersetzt, die auch in $data vorhanden sind).
Pro Wert in der Liste wird ein option-Tag mit value=”<value>” mit Inhalt <content> ausgegeben.
Bei jedem Wert, der dabei mittels dem Parameter ”selected” bzw. ”selected2” (siehe unten) als zu Beginn markiert erkannt wird, wird in seinem option-Tag das selected-Flag gesetzt.

selected: zu Beginn ausgewählter Wert, muss in $values vorhanden sein, d.h. er sollte als value-Komponente eines Elements im values-Array vorkommen.
Falls das multiple-Flag (siehe oben) gesetzt ist, kann dies eine Liste der zu Beginn ausgewählten Werte statt einem einzelnen Wert sein. Sonst sollte es nur ein einzelner Wert sein.

selected2: boolsches Flag; falls gesetzt, wird in jedem option-Tag statt dem festen String-Wert für das selected-Attribut ”selected” eine durch ´html_substitute_data´ später auswertbare bedingte Ausgabe für ”selected” ausgegeben. Dabei wird bei späterem Auswerten durch ´html_substitute_data´ ($subst_type & 8 == 1) nur dann tatsächlich der String ”selected” in einem option-Tag ausgegeben, wenn der zum option-Tag gehörende Wert (aus $values) gleich einem bei der späteren Auswertung an ´html_substitute_data´ übergebenen Vergleichswert ist.
Falls selected2 benutzt wird sollte selected nicht verwendet werden.
selected2 ist bei der Verwendung von Schablonen nützlich, d.h. man man erzeugt mit ´html_form´ eine Schablone für mehrere Formulare, derren tatsächliche Datensätze bei Aufruf von ´html_form´ noch nicht bekannt ist. Eine solcher Schablonen-Formular-Code kann z.B. im Parameter datacell_format des $param-Arrays für ´html_table´ verwendet werden, um für jedes in einer HTML-Tabelle angezeigte Datentupel einen eigenes Formular in der Tabellenzeile zu erzeugen (mittels Auswertung durch ´html_substitute_data´)

- type=”radio” oder type=”checkbox”: Formelement für Radio- bzw. Check-Box-Buttons (<input type=radio...> bzw <input type=checkbox...>); weitere Parameter in $e sind hierfür:

name: name der Radio/Checkbox, name-Attribut des input-Tags

value: Wert der Box, value-Attribut des input-Tags, wird htmlencoded

content: Beschriftung der Box, Inhalt des input-Tags, wird htmlencoded

content_htmlpre: Beschriftung der Box als HTML-Code, Inhalt des input-Tags, wird vor Wert von "content" ausgegeben

content_htmlpost: Beschriftung der Box als HTML-Code, Inhalt des input-Tags, wird nach Wert von "content" ausgegeben

content_html: Beschriftung der Box als HTML-Code, Inhalt des input-Tags; Alternative der Angabe der Beschriftung zu content/content_htmlpre/content_htmlpost

checked: boolsches Flag, falls true, so ist diese Box zu Beginn gecheckt, d.h. checked-Attribut des input-Tags gesetzt

checked2: boolsches Flag, falls gesetzt, wird im input-Tag statt dem festen String-Wert für das checked-Attribut "checked" eine durch ´html_substitute_data´ später auswertbare, bedingte Ausgabe von "checked" ausgegeben. Dabei wird bei späterem Auswerten durch ´html_substitute_data´ nur dann tatsächlich der String "checked" in im input-Tag ausgegeben, wenn der Wert der Box "value" gleich einem bei der späteren Auswertung an ´html_substitute_data´ übergebenen Vergleichswert ist.
Falls checked2 benutzt wird sollte checked nicht verwendet werden.
checked2 ist bei der Verwendung von Schablonen nützlich, d.h. man man erzeugt mit ´html_form´ eine Schablone für mehrere Formulare, derren tatsächliche Datensätze bei Aufruf von ´html_form´ noch nicht bekannt ist. Eine solcher Schablonen-Formular-Code kann z.B. im Parameter datacell_format des $param-Arrays für ´html_table´ verwendet werden, um für jedes in einer HTML-Tabelle angezeigte Datentupel einen eigenes Formular in einer Datenzeile zu erzeugen (mittels Auswertung durch ´html_substitute_data´).

- type="reset" oder type="submit": Formelement für Reset- bzw Submit-Button (<input type=reset...> bzw <input type=submit...>); weitere Parameter in $e sind hierfür:

  name: name des Buttons, d.h. name-Attribut des input-Tags

  value: Beschriftung des Buttons, d.h. value-Attribut des input-Tags, wird htmlencoded

- type="html": kein echtes Formelement, sondern Möglichkeit bei einem Aufruf von ´html_form´ bzw. ´html_form_elements´ beliebigen HTML-Code zwischen echte Formelemente zu setzten ohne einzelne Aufrufe zu ´html_form_element´ machen zu müssen. Einziger weiterer Parameter in $e ist:

  content: beliebiger HTML-Code, wird unmittelbar in Ausgabe übernommen

- type="constant": kein echtes Formelement, sondern sozusagen ein Readonly-Select-Menü: Verhält sich ähnlich wie select-Formelement bei der Ausgabe, allerdings wird der aktuelle Wert nur angezeigt und kein echtes Formelement aufgebaut, so dass vom HTML-Benutzer keine Änderung über diese Formelement möglich ist.
  Weitere Parameter in $e sind die gleichen wie bei type="select",
  Verwendung findet dieses Pseudoelement daher bei der Schablonenbenutzung, wo bei Aufbau des Formular-Quellcodes die tatsächlichen Werte des Datentupels noch nicht bekannt sind und später mittels ´html_substitute_data´ mit einem dann übergebenen Wert verglichen werden, womit dann entschieden werden kann, welcher Auswahlwert zu Beginn ausgewählt ist und damit angezeigt wird.

- bei allen Typen sind stets noch folgende Parameter in $e möglich:

htmlpre: beliebiger HTML-Code, der vor dem Formelement ausgegeben wird

htmlpost: beliebiger HTML-Code, der nach dem Formelement ausgegeben wird

$data: Ein gegebenes Datentupel, wobei ein Datentupel Datentupel ein assoziatives Array, bestehend aus beliebigen Key/Value-Paaren ist, z.b. wie es von db_query als Repräsentation einer Zeile eines SQL-Ergebnisses zurückgeliefert wird.

Rückgabe von ´html_form_element´ ist ein String, der den HTML-Quelltext für das Formelement enthält.

html_form_elements($elements, $data = array ()): Ähnlich wie es zur Funktion ´html_table_line´ die Funktion ´html_table_lines´ gibt, existiert zur Funktion ´html_form_elements´ eine Funktion die es ermöglicht mehrere Formelemente mit einem Funktionsaufruf zu erzeugen. Allerdings wird bei allen Formelementen stets nur Werte von einem fest vorgegebene Datentupel substituiert. Die Funktion heißt ´html_form_elements´.
Die Liste der Formelemente wird dabei vorgegeben. Die Werte eines einzigen Datentupels sind jedoch in diese Vorgabemaske substituierbar. Pro gegebenem Element wird ´html_form_element´ aufgerufen. Die Rückgabewerte der einzelnen Aufrufe, d.h. die HTML-Quelltexte der einzelnen Elemente, werden zu einem String zusammengesetzt und zurückgeliefert. Parameter sind (siehe auch ´html_form_element´):

$elements: Liste von Definitionen für Formelemente, d.h. jedes Listenelement wird an die Funktion ´html_form_element´ als Parameter $e übergeben. (siehe ´html_form_element´).

$data: Eine Liste gegebener Datentupel, wobei ein Datentupel Datentupel ein assoziatives Array, bestehend aus beliebigen Key/Value-Paaren ist, z.b. wie es von db_query als Repräsentation einer Zeile eines SQL-Ergebnisses zurückgeliefert wird.
Die Elemente von $data werden einzeln als Parameter $data an ´html_form_element´ übergeben.

Funktionsrückgabe ist ein String, der den HTML-Quelltext für die Formelemente enthält.

html_form_end():

Diese Funktion erzeugt (ähnlich wie ´html_table_end´) HTML-Code für das Ende eines Formulares. (</input>-Tag). Rückgabe ist ein String mit diesem HTML-Code. Hiermit wird ein durch ´html_form_head´, ´html_form_elements´ bzw. ´html_form_element´ begonnenes Formular abgeschlossen.

html_form($param, $elements, $data = array ()): Um den HTML-Quelltext für ein ganzes Formular, d.h. den Formularbeginn, die Elemente des Formulares und das Formular-Ende zu erzeugen, dient diese Wrapper-Funktion (ähnlich ´html_table´). Die Liste der Elemente wird dabei vorgegeben. Die Werte eines vorgegebenen Datentupels sind jedoch in diese Vorgabemaske substituierbar. Pro gegebenem Element wird ´html_form_element´ aufgerufen. Der Formular-Beginn bzw. das Ende werden von ´html_form_head´ bzw. ´html_form_end´ erstellt. Die Rückgabewerte der einzelnen Aufrufe, d.h. die HTML-Quelltexte der einzelnen Elemente, werden zu einem String zusammengesetzt und zurückgeliefert. Parameter sind (siehe auch ´html_form_head´, ´html_form_element´, ´html_form_end´):

$param: Array von möglichen Parametern für den Formularbeginn, siehe ´html_form_head´

$elements: Liste von Definitionen für Formelemente, d.h. jedes Listenelement wird an die Funktion ´html_form_element´ als Parameter $e übergeben

$data: Ein gegebenes Datentupel, wobei ein Datentupel Datentupel ein assoziatives Array, bestehend aus beliebigen Key/Value-Paaren ist, z.b. wie es von ´db_query´ als Repräsentation einer Zeile eines SQL-Ergebnisses zurückgeliefert wird, d.h. die Elemente von $datas werden einzeln als Parameter $entry an ´html_form_element´ übergeben.

Rückgabe der Funktion ist ein String, der den HTML-Quelltext für das gesamte Formular enthält.

Funktionen für die Erstellung anderer HTML- Elemente:

html_paragraph($htmltext): Diese Funktion erzeugt HTML-Code für einen Absatz (p-Tag) um einen gegebenem HTML-Code als Inhalt. Sie hat nur folgenden Parameter:

$htmltext: beliebiger HTML-Text als Inhalt für den Absatz

Rückgabe ist ein String mit dem HTML-Code für den Absatz.

html_div($htmltext, $alignment="center"): Um gegebenen HTML-Code in einer Gruppierung (div-Tag) einzuschließen dient diese Funktion. Es gibt folgende Parameter:

$htmltext: beliebiger HTML-Text als Inhalt für die Gruppierung

$alignment: Ausrichtung der Gruppierung (align-Attribut des div-Tags)

Rückgabe ist ein String mit HTML-Code für die Gruppierung.

html_nobreak($htmltext): Diese Funktion umschließt gegebenen HTML-Code mit nobr-Tag, um bei der HTML-Darstellung Zeilenumbrüche zu vermeiden. Es gibt nur einen Parameter:

$htmltext: beliebiger HTML-Text als Inhalt

Rückgabe ist String mit HTML-Code, bei dem keine Zeilenumbrüche bei der Darstellung auftreten sollen.

html_list($listtype, $items, $list_param = array(), $caption = ""): Um eine HTML-Liste zu erzeugen, gibt es die Funktion 'html_list'. Dabei gibt der Parameter $listtype den Listen-Typ an und hat als möglich Werte "ordered" (ol, li, lh-Tag), "unordered" (ul, li, lh-Tag) oder "definition" (dl, dt, lh-Tag). Insgesamt gibt es diese Parameter:

$listtype: Type der Liste: "ordered", "unordered" oder "definition"

$items: Liste von HTML-Texten, die als Inhalt für die HTML-Listen-Elemente verwendet werden

$list_param: Array von Parametern für die Liste:

- type: type-Attribut des Listen-Tags (ol, ul bzw. dl): bei $listtype="ordered" z.B. "a","A","I","i", sonst z.B. "circle", "square" oder "disc"
- start: start-Attribut des ol-Attributs bei geordneten Listen
- compact: boolsches Flag, falls gesetzt wird compact-Attribut im Listen-Tag (ol, ul bzw. dl) gesetzt.
- empty_caption_allowed: boolsches Flag; falls gesetzt, wird stets eine Überschrift angezeigt, auch wenn sie (Parameter $caption) leer ist (d.h. "<dl></dl>" taucht im HTML-Code auf)

$caption: mögliche Überschrift der Liste (lh-Tag)

Rückgabe ist ein String mit dem HTML-Codefür die HTML-Liste.

Für jeden Listentyp existiert eine Wrapperfunktion:

html_ordered_list($items, $list_param = array(), $caption = ""): Diese Funktion erzeugt HTML-Code für nummerierte HTML-Liste. Dazu gibt sie ihre eigenen Parameter zusammen mit $listtype="ordered" an die Funktion 'html_list' weiter und gibt derren Ergebnis zurück. Parameter sind (siehe 'html_list'):

$items: Liste von HTML-Texten, die als Inhalt für die HTML-Listen-Elemente verwendet werden

$list_param: Array von Parametern für die Liste, siehe 'html_list'

$caption: mögliche Überschrift der Liste, siehe 'html_list'

Rückgabe ist ein String mit HTML-Code der nummerierten HTML-Liste.

html_unordered_list($items, $list_param = array(), $caption = ""): Mit dieser Funktion wird HTML-Code für eine unnummerierten HTML-Liste erzeugt. Sie gibt ihre eigenen Parameter zusammen mit $listtype="unordered" an die Funktion 'html_list' weiter und gibt derren Ergebnis zurück. Als Parameter hat sie (siehe 'html_list'):

$items: Liste von HTML-Texten, die als Inhalt für die HTML-Listen-Elemente verwendet werden

$list_param: Array von Parametern für die Liste, siehe 'html_list'

$caption: mögliche Überschrift der Liste, siehe 'html_list'

Rückgabe ist ein String mit HTML-Code der unnummerierten HTML-Liste.

html_definition_list($items, $list_param = array(), $caption = ""): Wrapperfunktion für 'html_list', die HTML-Code für eine HTML-Definitions-Liste erstellt. Dafür gibt diese Funktion ihre eigenen Parameter zusammen mit $listtype="definition" an 'html_list' weiter und gibt derren Ergebnis-String zurück. Sie hat folgende Parameter (siehe 'html_list'):

$items: Liste von HTML-Texten, die als Inhalt für die HTML-Listen-Elemente verwendet werden

$list_param: Array von Parametern für die Liste, siehe 'html_list'

$caption: mögliche Überschrift der Liste, siehe 'html_list'

Rückgabe ist ein String mit HTML-Code der Definitions-HTML-Liste.

Auch zum Erstellen von HTML-Links gibt es einige Funktionen:

html_link($url, $text, $options = ""): Diese Funktion erzeugt HTML-Code für einen Link aus gegebener URL, gegebenem nicht-HTML-kodiertem Anzeigetext, sowie einem String, der ggf. Attributdefinitionen für den Link enthält. Der gegebene Anzeigetext wird htmlencoded und dann in den Link eingebaut. Dies sind die Funktions-Parameter:

$url: URL des Link, d.h. href-Attribut des a-Tags

$text: Textstring als Beschriftung, d.h. Inhalt des a-Tags, wird vorher htmlencoded

$options: String der evtl. Attributdefinitionen für den a-Tag enthält

Rückgabe der Funktion ist ein String des HTML-Code für den Link.

html_img($param): Diese Funktion erstellt HTML-Code ein Bild (img-Tag). Parameter sind:

$param: Array von Parametern für das Bild:

- src: URL des Bildes, bzw. src-Attribut des img-Tags
- alt: alternativer Text, falls Bild-Datei nicht vorhanden/verwendbar bzw. alt-Attribut des img-Tags; der Text wird vor Benutzung htmlencoded
- width: Weite des Bildes bzw. width-Attribut des img-Tags
- height: Höhe des Bildes bzw. height-Attribut des img-Tags
- border: Randstärke? des Bildes bzw. border-Attribut des img-Tags

Rückgabe von 'html_img' ist ein String, der HTML-Code für das Bild enthält.

html_link_with_image($url, $img_param, $text = "", $options = ""): Diese Funktion produziert mit Hilfe von 'html_img' und 'html_link' HTML-Code für einen Link mit Bild aus gegebener URL, gegebener Bildinformation, gegebenem nicht-HTML-kodiertem Anzeigetext sowie einem String, der ggf. Attributdefinitionen für den Link enthält. Der Anzeigetext wird htmlencoded und dann in den Link eingebaut. Der Teilstring für das Bild wird von 'html_image' aus $img_param erzeugt. Parameter sind:

$url: URL des Link, d.h. href-Attribut des a-Tags

$img_param: Parameter-Array für das Bild, siehe 'html_image'; z.B. legt der Parameter src die URL und der Parameter alt den alternativen Text des Bildes fest.

$text: Textstring als Beschriftung, d.h. Inhalt des a-Tags, wird vorher htmlencoded

$options: String der evtl. Attributdefinitionen für den a-Tag enthält

Rückgabe ist ein String, der den HTML-Code des Links mit Bild enthält.

html_link_raw($url, $htmltext, $options = ""): Ähnlich wie 'html_link' erzeugt HTML-Code für einen Link. Allerdings muss der gegebene Text hier bereits htmlencoded sein. Parameter sind (vergleiche 'html_link'):

$url: URL des Link, d.h. href-Attribut des a-Tags

$htmltext: beliebiger HTML-Text als Beschriftung, d.h. Inhalt des a-Tags

$options: String der evtl. Attributdefinitionen für den a-Tag enthält

Rückgabe ist ein String für den HTML-Code des Links.

html_headline($text, $level = 2): Um HTML-Code für eine Überschrift zu erzeugen, dient 'html_headline'. Die Parameter für diese Funktion sind:

$text: Überschriftstext, d.h. Inhalt des Überschrifts-Tags; wird vor Benutzung htmlencoded

$level: Größe der Überschrift, d.h. verwendeter h-Tag (h1, h2, h3, ...), Standart 2

Rückgabe ist ein String, der den HTML-Text der Überschrift enthält.

html_textattribs($htmltext, $type="bold"): Diese Funktion umschließt gegebenen HTML-Text mit einem gegebenen Tag für Text-Formatierung (logisch oder physisch), wie z.B. "bold", "italic", etc. Parameter sind die folgenden:

$htmltext: gegebener HTML-Text

$type: Art der Text-Formatierung bzw Tag-Name: mögliche Werte sind "bold" (b-Tag), "italic" (i-Tag), "underline" (u-Tag), "strike" (s-Tag), "big" (big-Tag), "small" (small-Tag), "subscript" (sub-Tag), "teletyper" (tt-Tag), "blink" (blink-Tag), "strong" (strong-Tag), "emphasize" (em-Tag), "cite" (cite-Tag), "variable" (var-Tag), "keyboard" (kbd-Tag), "sample" (samp-Tag), "definition" (dfn-Tag), "code" (code-Tag) oder Name eines beliebigen anderen Tags

Rückgabe ist ein String, der HTML-Code mit der Text-Formatierung enthält.

html_fgcolor ($htmltext, $fgcolor): Diese Funktion setzt die Vordergrundfarbe um einen gegebenen HTML-Code, d.h. umschließt ihn mit font-Tag (<font color=...>). Parameter sind:

$htmltext: gegebener HTML-Code, dessen Vordergrundfarbe gesetzt werden soll

$fgcolor: gegebene HTML-Farbspezifikation

Rückgabe ist ein String, der in gegebenem HTML-Code Vordergrundfarbe gesetzt hat.

html_nl($count = 1): Um HTML-Code für eine gegebene Anzahl von Zeilenumbrüchen (br-Tag) zu erstellen ist 'html_nl' gedacht. Folgendes ist der einzige Parameter der Funktion:

$count: Anzahl der Zeilenumbrüche, Standart 1

Rückgabe ist ein String, der HTML-Code für die Zeilenumbrüche enthält.

html_space($count = 1): Diese Funktion erzeugt HTML-Code für gegebene Anzahl von Leerzeichen ( ). Parameter ist nur:

$count: Anzahl Leerstellen, Standart 1

Rückgabe ist ein String, der HTML-Code für die Leerstellen enthält.

html_comment($text): Mit dieser Funktion kann HTML-Code für HTML-Kommentar (<!– ... –>) mit gegebenem Text generiert werden. Parameter ist nur:

$text: beliebiger Kommentartext, d.h. Inhalt des <!– ... –>-Tags.

Rückgabe ist ein String, der HTML-Code für den Kommentar enthält.

html_comment2($text): Wie 'html_comment' erzeugt diese Funktion einen HTML-Kommentar. Der Kommentar-Text wird aber mit einer Anzahl von +-Zeichen aufgefüllt, um eine deutlichere Trennung im Quelltext hervorzurufen. Außerdem werden vor dem Kommentar im Quelltext 2 ASCII-Zeilenumbrüche eingebaut. Parameter ist nur.

$text: beliebiger Kommentartext

Rückgabe ist ein String, der HTML-Code für den Kommentar enthält.

string2html($text): Diese Funktion wandelt einen beliebigen, gegebenen Text in HTML, d.h. macht daraus einen htmlencodeten String. D.h. z.B. werden Umlaute oder “<”, “>”, etc. entsprechend kodiert. Parameter ist dabei:

$text: beliebiger Eingabe-String

Rückgabe ist der htmlencodete String.

html_link_parameter ($key, $val): Um aus einem gegebenen Key/Value-Paar in einen String für eine eine Parameterübergabe per Link zu erstellen, gibt es diese Funktion. Parameter sind die folgenden:

$key: Schlüsselname des Parameters, d.h. später per Link übergebener Parameter bzw. späterer php-Variablenname nach Aufruf des vollständigen Links

$value: Wert des Parameters

Rückgabe ist der Teil-String für Parameterübergabe von <Key>=<Value> per Link.

html_start($title, $body_param = array (), $headparam = array()): Mit dieser Funktion wird der HTML-Code für den Beginn einer HTML-Seite, d.h. erzeugt Code für den eigentlichen HTML-Beginn, HTML-Kopf und den Beginn des HTML-Bodies, erstellt. Parameter sind:

$title: Überschrift der HTML-Seite bzw. Inhalt des title-Tags des head-Tags, wird htmlencoded

$body_param: Parameter-Array für den HTML-Body: enthält selbst bisher nur den Parameter ”bgcolor” für den Wert des bgcolor-Attributs des body-Tags

$head_param: Parameter-Array für den HTML-Kopf: enthält selbst bisher nur den Parameter “stylesheet_link”, der die URL für ein Stylesheet, d.h. href-Attribut eines <link rel=stylesheet...>-Tags, angibt

Rückgabe ist ein String, der HTML-Code für den Seitenbeginn enthält.

html_stop(): Diese Funktionbildet das Gegenstück zu ’html_start’. Sie erstellt Code für das Ende einer HTML-Seite, d.h. für das Ende des HTML-Bodies (</body>) und des Endes des HTML-Quellcodes (</html>). Sie hat keine Parameter.

Rückgabe ist ein String, der HTML-Code für das HTML-Ende enthält.

html_usertext($usertext, $type, $default = ””): Mit dieser Funktion wird ggf. HTML-Code erzeugt, um einen vorgegebenen Text anzuzeigen. Dies dient vor allem für Beschreibungen, Erklärungen etc. zwischen andere HTML-Elemente zu setzen. Das Parameter-Array $usertext kann dabei außer den hier verarbeiteten Parametern viele andere - z.B. weitere usertext-Anzeige-Parameter für einen anderen Wert von $type - enthalten. Diese Funktion dient dazu, die durch $type spezifizierten Parameter aus $usertext zu filtern und die darin enthaltene Meldung anzuzeigen. Sind die durch $type bestimmten Parameter in $usertext nicht vorhanden, so wird nichts angezeigt oder $default angezeigt, falls spezifiziert. Folgendes sind die Parameter der Funktion:

$type: vorgegebener Schlüsselname, um verschiedene Beschreibungstexte in einem Parameter-Array zu unterscheiden, siehe Parameter $usertext

$usertext: Array von Parametern, die den anzuzeigenden Text beschreiben. Es gibt 2 mögliche Parameter, beide abhängig von $type:

- ”html$type”, d.h. der String ”html” und daran angehängt $type:

beliebiger HTML-Text der ausgegeben wird

- "text$type", d.h. der String "text" und daran angehängt $type:

beliebiger Text der htmlencoded wird und in einen Paragraph (p-Tag) eingeschlossen ausgegeben wird

$default: Standardwert für den Text, falls "html$type" bzw. "text$type" leer bzw. nicht angegeben sind

Rückgabe ist ein String, der HTML-Code für die Meldung, falls eine angegeben wurde, enthält.

Funktionen zur Erzeugung von Links/Hidden-Formelementen zur HTTP-Übergabe von Arrays: Um per HTML-Link oder per Formular mit Hidden-Formelementen ein ganzes Array rekursiv an eine neu aufgerufene Instanz eines php-Skriptes zu übergeben dienen folgende Funktionen:

html_link_parameter_array ($name, $data): Diese Funktion wandelt ein gegebenes Array rekursiv in einen String um für eine Parameterübergabe per Link um. Die Werte des Arrays werden dabei urlencoded. Zum Bestimmen aller Key/Value-Paare des Arrays, wird die Hilfsfunktion 'html_arrayquery_parameter_pairs' verwendet. Die Parameter dieser Funktion sind:

$name: Schlüsselname des Parameter-Arrays, d.h. später per Link übergebener Parameter bzw. späterer Variablenname bei Aufruf des vollständigen Links

$data: gegebenes Array, ein Wert $data[key1][key2]...[keyn] erhält dabei im Parameterstring den Schlüsselnamen "$name[key1][key2]...[keyn]". PHP erkennt dies beim Aufruf durch den Link als den Wert für den Index "[key1][key2]...[keyn]" der Arrayvariable mit Namen $name.

Die Rückgabe ist String für Parameterübergabe des Arrays per Link, d.h. ein String bestehend aus einer Folge von "<name>=<urlencodeter value>", getrennt durch "&". Dieser kann als Teil eines Links verwendet werden.

html_link_parameter_array_raw ($name, $data): Diese Funktion arbeitet ähnlich wie 'html_link_parameter_array'. Allerdings werden die Werte des gegebenen Arrays $data nicht urlencoded, d.h. sollten es bei Aufruf bereits sein, bzw. später noch bearbeitet werden.
Dies wird z.B. verwendet, wenn die Werte von $data Variablenbezeichner enthalten, die durch 'html_substitute_data' später noch ersetzt werden sollen, d.h. die Rückgabe dieser Funktion als Schablone für mehrere Linkaufrufe benutzt werden soll. 'html_link_parameter_array' würde ja die Variablenbezeichner entsprechend urlencoden und damit für 'html_substitute_data' unkenntlich machen. (siehe 'html_substitute_data').
Parameter sind folgende (Vergleiche 'html_link_parameter_array'):

$name: Schlüsselname des Parameter-Arrays, d.h. später per Link übergebener Parameter bzw. späterer Variablenname bei Aufruf des vollständigen Links

$data: gegebenes Array, ein Wert $data[key1][key2]...[keyn]erhält dabei im Parameterstringden Schlüsselnamen "$name[key1][key2]...[keyn]".PHP erkennt dies beim Aufruf durch den Link als den Wert für den Index "[key1][key2]...[keyn]" der Arrayvariable mit Namen $name.

Die Rückgabe ist String für Parameterübergabe des Arrays per Link, d.h. ein String bestehend aus einer Folge von "<name>=<urlencodeter value>", getrennt durch "&". Dieser kann als Teil eines Links verwendet werden.

html_form_parameter_array ($name, $data): Um ein ganzes Array nicht per Link, sondern per Hidden-Elementen eines Formulares, an eine neue Skript-Instanz zu übergeben, dient diese Funktion. Sie erzeugt aus gegebenem Array mehrere Hidden-Form-Elemente um das Array rekursiv per Formular zu übertragen. Dabei verwendet sie die Funktion 'html_form_element' und die Hilfsfunktion 'html_arrayquery_parameter_pairs'. Parameter sind (Vergleiche 'html_link_parameter_array'):

$name: Schlüsselname des Parameter-Arrays, d.h. später über ein oder mehrere Hidden-Formelemente übergebene Parameter bzw. späterer Variablenname bei Abschicken des vollständigen Formulares.

$data: gegebenes Array, ein Wert $data[key1][key2]...[keyn] wird dabei in einem eigenen Hidden-Formelement mit Namen (name=) "$name[key1][key2]...[keyn]" abgelegt. Die Werte werden dabei nicht htmlencoded.

Rückgabe ist ein String für Parameterübergabe des Arrays durch Hidden-Form-Elemente, d.h. ein String der HTML-Code für mehrere Hidden-Form-Elemente enthält. Um ein vollständiges Formular zu erhalten muss davor bzw. dahinter der Beginn bzw. das Ende eines HTML-Formulares - z.B. durch 'html_form_head' bzw. 'html_form_end' erzeugt - gesetzt werden.

***html_arrayquery_parameter_pairs($prefix, $data):** Dies ist nur eine Hilfsfunktion für 'html_form_parameter_array', 'html_link_parameter_array', 'html_link_parameter_array_raw'. Sie erzeugt Parameter-Namens-String für HTML-Links oder Hidden-Formelemente aus dem gegebenen Array $data (rekursiv). Sie hat folgende Parameter:

$prefix: Präfix des Schlüsselnamens des Parameter-Arrays

$data: gegebenes Array

Rückgabe ist eine Liste von Schlüsselnamen. Jeder Arraywert $data[key1][key2]...[keyn] wird dabei zu einem Schlüsselnamen "$prefix[key1][key2]...[keyn]".

Funktion zum Ersetzen von Variablenwerten in Strings: Um v.a. beim Erstellen von Tabellen und Formularen das Einsetzen von Daten aus Datentupeln, wie sie z.B. aus der Datenbank geliefert werden, in vorgegebene Stellen, wie z.B. Inhalt einer Tabellenspalte oder Wert eines Formularfeldes, zu ermöglichen gibt es die Funktion ´html_substitute_data´:

***html_substitute_data ($string, $data, $allflag = 0, $subst_type = 7, $defaultkey = "", $complex_names_allowed = 0, $startchar = '$'):** ´html_substitute_data´ ersetzt in einem gegebenem Eingabestring $string bestimmte Strings durch andere. Vor allem wird die Ersetzung von Variablenbezeichnern durch Variablenwerte aus dem gegebenen Datentupel $data damit realisiert. Weiterhin besteht die Möglichkeit sehr einfache Bedingungen auszuwerten.
Es gibt 5 verschiedene Arten von Ersetzungen (Variablenname bei den Punkten 1. bis 3. meint im folgenden einen String, der nur aus Klein/Groß-Buchstaben, Zahlen und den Zeichen "-" und "_" besteht. Ist $complex_names_allowed true, so sind auch noch die Zeichen "[" und "]" zugelassen):

1. String "$(keyname)", wobei keyname ein Variablenname ist, wird ersetzt durch Wert des Datentupels $data[keyname] bzw.
   ist der String $defaultkey nicht leer, so werden die Strings "$()" und "$$" beide jeweils ersetzt durch den Wert des Datentupels $data[$defaultkey], d.h. bei fehlendem keyname wird $defaultkey verwendet;
   Die Ersetzung findet jeweils nur statt, falls $data[keyname] definiert ist oder $allflag true ist.
   Diese Ersetzung findet außerdem nur dann statt, wenn Bit 0 von $subst_type gesetzt ist.

2. Ähnlich zu 1.: der String "$[keyname]", wobei keyname ein Variablenname ist, wird ersetzt durch den htmlencodeten (z.B. Umlaute, "<",">",... codieren) Wert des Datentupels $data[keyname] bzw.
   ist der String $defaultkey nicht leer, so wird der String "$[]" ersetzt durch den htmlencodeten Wert des Datentupels $data[$defaultkey], d.h. bei fehlendem keyname wird $defaultkey verwendet;
   Die Ersetzung findet nur statt, falls $data[keyname] definiert ist oder $allflag true ist.
   Außerdem findet diese Ersetzung nur statt, wenn Bit 1 von $subst_type gesetzt ist.

3. Ähnlich zu 1.: der String "$<keyname>", wobei keyname ein Variablenname ist, wird ersetzt durch den urlencodeten (für Verwendung in Parameterstrings in URLs) Wert des Datentupels $data[keyname] bzw.
   ist der String $defaultkey nicht leer, so wird der String "$<>" ersetzt durch den urlencodeten Wert des Datentupels $data[$defaultkey], d.h. bei fehlendem keyname wird $defaultkey verwendet;
   Die Ersetzung findet nur statt, falls $data[keyname] definiert ist oder $allflag true ist.
   Außerdem wird nur dann ersetzt, falls Bit 2 von $subst_type gesetzt ist.

4. Falls der String $defaultkey nicht leer ist, wird der String
   "$?%(<urlencodeter Vergleichswert>:<Ausgabe>)" durch leeren String oder durch den String "<Ausgabe>" ersetzt, je nachdem, ob der urlencodete Wert $data[$defaultkey] ungleich oder gleich dem String "<urlencodeter Vergleichswert>" ist. Dabei ist "<urlencodeter Vergleichswert>" ein urlencodeter String und "<Ausgabe>" ein beliebiger String, der kein ")" enthält. Dies wird v.a. bei den Form-Elementen select, radio und checkbox bei Benutzung des selected2 bzw. checked2-Flags ausgenutzt (siehe ´html_form_element´).
   Diese Ersetzung findet nur statt, falls Bit 3 von $subst_type gesetzt ist.

5. Ein String der Form "$?<bedingung>(<attributname>:<Ausgabe>)", wobei der String "<bedingung>" nur aus Buchstaben besteht sowie der String "<attributname>" kein ":" und kein ")" und der String "<Ausgabe>" kein ")" enthalten darf, wird ersetzt durch den leeren String oder durch "<Ausgabe>", je nachdem, ob die durch <bedingung> und <attributname> spezifizierte Bedingung wahr ist oder nicht.
   Zur Zeit sind folgende Bedingungen möglich:

- "<bedingung>"="NULL": Die Bedingung ist wahr, falls der Wert des Datentupels $data[<attributname>"] gleich dem leeren String ist,

- "<bedingung>"="NOTNULL": Die Bedingung ist wahr, falls der Wert des Datentupels $data[<attributname>"] ungleich dem leeren String ist

  Diese Ersetzung findet nur statt, falls Bit 4 von $subst_type gesetzt ist.

Der Eingabe-String wird von links nach rechts nach den zu ersetzenden Strings durchsucht und ggf. Ersetzungen vorgenommen. Insbesondere wird ein Teilstring, der durch Ersetzung entstanden ist nicht noch einmal neu durchsucht. Sonst wären unter Umständen unerwünschte Ersetzungen die Folge. Denn dann könnte z.B. ein Variablenbezeichner der im Inhalt einer Variable, d.h. im Datentupel steht (was ja z.B. aus der Datenbank stammen kann und beliebige Strings enthalten kann) selbst ersetzt werden, oder anders gesagt, wären die Variablenbezeichner nie als echter Dateninhalt möglich.

´html_substitute_data´ war ursprünglich nur als Hilfsfunktion zu den Tabellen- bzw. Form-Erzeugungsfunktionen gedacht. Sie wird jedoch auch von einigen anderen Teilen des Workflowsystem, z.B. db-table-Funktionen benutzt.
Sie hat die folgende Parameter:

$string: vorgegebener String, in dem ggf. Ersetzungen durchzuführen sind

$data: gegebenes Datentupel, wobei ein Datentupel ein assoziatives Array ist, bestehend aus beliebigen Key/Value-Paaren ist, z.b. wie es von db_query als Repräsentation einer Zeile eines SQL-Ergebnisses zurückgeliefert wird

$allflag: boolsches Flag, falls true werden auch Variablenersetzungen vorgenommen, für die im Datentupel kein Wert existiert, d.h. der Variablenbezeichner wird dann durch den leeren String ersetzt und verschwindet damit aus dem String
Wird z.B. Bei der Tabellenerzeugung verwendet.

$subst_type: Bitvektor der angibt, welche Art von Ersetzung erlaubt ist. Dabei wird eine Ersetzung erlaubt, falls entspr. Bit gesetzt (Vgl. Liste der Ersetzungen oben):

Bit 0 (1): Typ-1-Ersetzung, z.B. “$(keyname)” -> $data[keyname]

Bit 1 (2): Typ-2-Ersetzung, z.B. “$[keyname]” -> htmlencoded $data[keyname]

Bit 2(4): Typ-3-Ersetzung, z.B. “$<keyname>” -> urlencoded $data[keyname]

Bit 3(8): Typ-4-Ersetzung, Vergleich von $data[$defaultkey] mit festem Wert

Bit 4 (16): Type-5-Ersetzung, NOT/NOTNULL-Bedingung für ein Wert von $data

$defaultkey: Name des Standardattributs für das Datentupel $data, Verwendung siehe Liste der Ersetzungen oben

$complex_names_allowed: boolsches Flag, falls true sind in Variablennamen der Ersetzungen ”$(keyname)”, ”$[keyname]” und ”$<keyname>” (Typ 1.-3.) außer Buchstaben, Zahlen, ”-” und ”_” auch ”]” und ”[” zugelassen

$startchar: ggf. statt dem Zeichen ’$’ anderes Zeichen, das möglichen Ersetzungspunkt im Eingabestring anzeigt, d.h. in den obigen Ausführungen in allen Strings ”$” durch dieses neue Zeichen ersetzen;
ist z.B. $startchar=”H”, so lauten die zu ersetzenden Strings nicht “$(keyname)”, “$()”, “$$”, “$[keyname]”, “$[]”, “$<keyname>”, “$<>”, “$?%(<urlencodeter Vergleichswert>:<Ausgabe>)” und “$?<bedingung>(<attributname>:<Ausgabe>)”, sondern ”H(keyname)”, ”H()”, ”HH”, ”H[keyname]”, ”H[]”, ”H<keyname>”, ”H<>”, ”H?%(<urlencodeter Vergleichswert>:<Ausgabe>)” und”H?<bedingung>(<attributname>:<Ausgabe>)”

Rückgabe ist der Eingabe-String nach der Ersetzung.

4.6.3 Funktionen für Erzeugung von Debugmeldungen

Die folgenden Funktionen dienen der Erzeugung von Debug-Meldungen. Im Gegensatz zu allen anderen Funktionen in ´lib-html´ geben sie keinen String als Rückgabe, sondern geben selbst die Debugmeldung auf Stdout aus.

debug($text, $level=1, $file=””, $kennzeichen=”DEBUG: ”): Diese Funktion gibt HTML-Code für eine Debug-Meldung aus. Parameter sind:

$text: Debug-text, wird htmlencoded und in ”bold” (bold-Tag) ausgegeben

$level: Debug-Level, nur Meldungen die ein Level <= dem der globalen Variable $debuglevel haben, werden ausgegeben

$file: falls globale Variable $debugfiles eine Liste von (File)namen enthält, so wird eine Meldung nur ausgegeben, falls der Name $file darin enthalten ist

$kennzeichen: Anzeige-Präfix, um Debug zu kennzeichnen

debug_raw($htmltext, $level=1, $file=""): Im Gegensatz zu ´debug´ muss bei ´debug_raw´ die Debug-Meldung bereits htmlencoded vorliegen. Dafür kann sie aber beliebiger HTML-Code sein.

$text: Debug-text als HTML-Code

$level: Debug-Level, nur Meldungen die ein Level <= dem der globalen Variable $debuglevel haben, werden ausgegeben

$file: falls globale Variable $debugfiles eine Liste von (File)namen enthält, so wird eine Meldung nur ausgegeben, falls $file darin enthalten ist

debug_array($prefix, $data, $level=1, $file="", $kennzeichen="DEBUG: "): Diese Funktion erzeugt HTML-Code, um Inhalt eines Arrays rekursiv, zeilenweise als Debugmeldungen auszugeben. Verwendet wird die Funktion ´debug´. Parameter sind:

$prefix: Text der vor den Array-werten angezeigt wird

$data: gegebenes, beliebiges Array

$level: Debug-Level, nur Meldungen die ein Level <= dem der globalen Variable $debuglevel haben, werden ausgegeben

$file: falls globale Variable $debugfiles eine Liste von (File)namen enthält, so wird eine Meldung nur ausgegeben, falls $file darin enthalten ist

$kennzeichen: Anzeige-Präfix, um Debug zu kennzeichnen

debug_list($text, $data, $level=1, $file="", $kennzeichen="DEBUG: "): Diese Funktion erzeugt HTML-Code, um Inhalt einer Liste in einer Zeile als Debugmeldung auszugeben. Verwendet wird dazu die Funktion ´debug´. Parameter sind:

$prefix: Text der vor den Listen-Werten angezeigt wird

$data: gegebene, beliebige Liste

$level: Debug-Level, nur Meldungen die ein Level <= dem der globalen Variable $debuglevel haben, werden ausgegeben

$file: falls globale Variable $debugfiles eine Liste von (File)namen enthält, so wird eine Meldung nur ausgegeben, falls $file darin enthalten ist.

$kennzeichen: Anzeige-Präfix, um Debug zu kennzeichnen

4.7 TGI als Beispiel für einen Workflow

Für die Anforderungsanalyse wurden Workflows für die Vorlesung Technische Grundlagen der Informatik, für das Rechnernetzepraktikum, für ein Fopra und eine Diplomarbeit erstellt und analysiert. Der Workflow zu TGI (siehe Abbildung 9) wurde nach der Implementierung der Anwendung als Beispiel realisiert und nach der Einrichtung des Systems in der Praxis getestet. Der Workflow besteht aus 10 Schritten, wobei der Schritt "Klausur" einen Sub-Workflow mit seinerseits 11 Schritten enthält. Der Subworkflow wurde eingerichtet, da mehrere Klausuren abgehalten wurden. Im folgenden werden die einzelnen Workflow-Schritte kurz beschrieben.

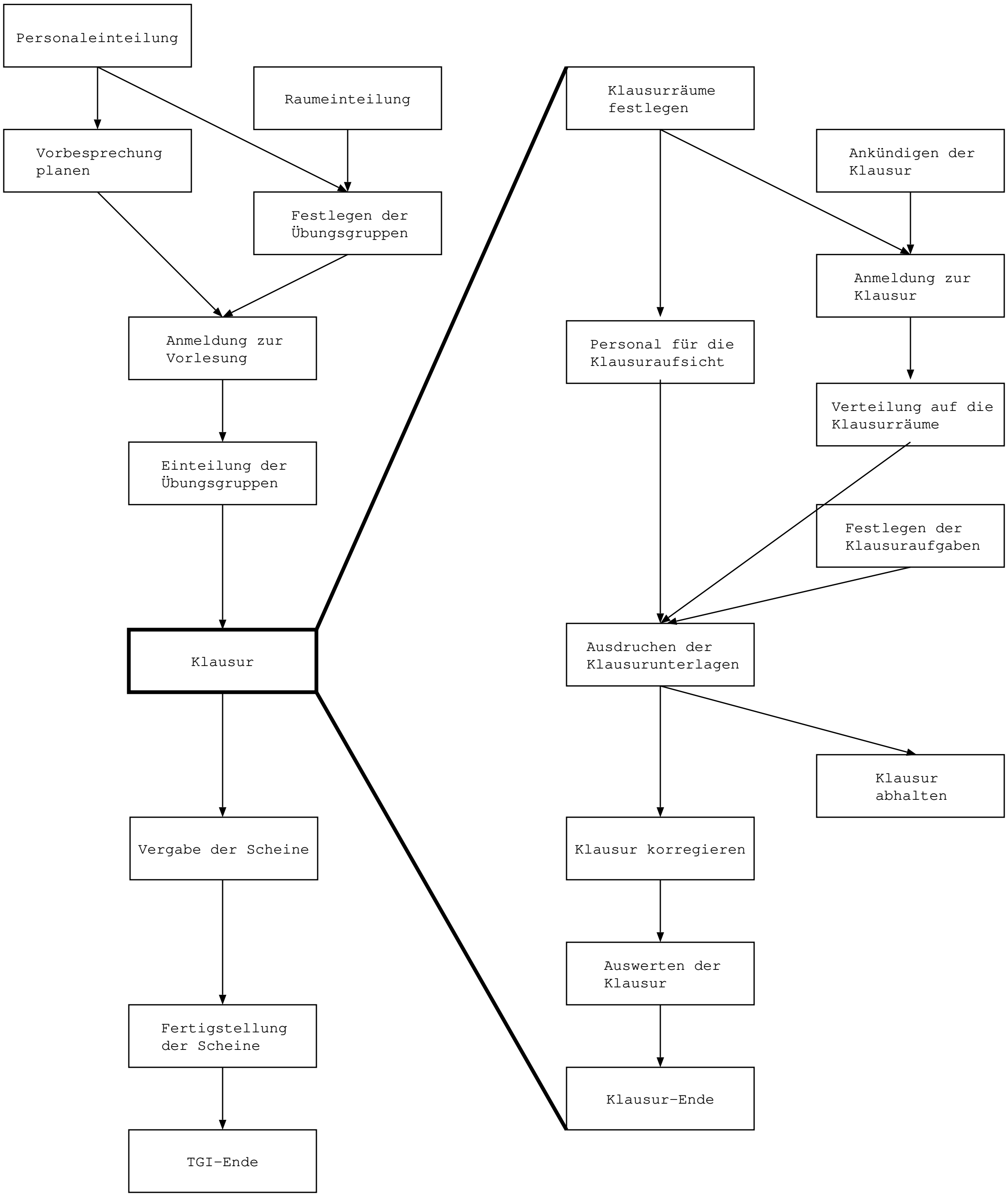

Abbildung 9: Der TGI-Workflow

1. Der Workflow-Schritt “Personaleinteilung” nutzt die Modul-Skripte ’module_Personal.inc.php’ und ’modul_SQLBedingung’. Letzteres Ermöglicht die Angabe mehrerer SQL-Bedingungen, die erfüllt sein müssen, bevor der Schritt als erledigt gekennzeichnet werden kann. In diesem Schritt wird überprüft, ob in der Relation ’arbeitet_für_vorlesung’ ein Tupel für diese Vorlesung eingetragen wurde. Das erste Modul-Skript enthält drei DBTable-Submodule. Das erste stellt das Personal für die Vorlesung dar. Die beiden anderen geben eine Liste der Mitarbeiter bzw. Studenten wieder. Aus diesen Listen können Personen ausgewählt werden, die dann zum Vorlesungspersonal hinzugefügt werden.

2. Im Schritt “Raumeinteilung” wird wieder das Modul für SQL-Bedingungen verwendet. Hier wird geprüft, ob ein Raum für die Vorlesung (Relation ’vorlesung_findet_statt_in’) reserviert wurde. Das Modul ’module-Raum.inc.php’ besteht aus zwei DBTable-Submodulen, eines für die Liste der reservierten Räume, eines für eine Liste aller Räume. Aus der zweiten Lisete können Räume ausgewählt werden, die dann zu den reservierten Räumen hinzugefügt werden. In der Tabelle der reservierten Räume kann der Wochentag, die Uhrzeit und Belegungsdauer festgelegt werden.

3. Der Schritt “Vorbesprechung planen” ist ein reiner Informationsschritt, der nur einen Text ausgibt. Es wird kein Modul eingebunden.

4. Der Schritt “Festlegen der Übungsgruppen” verwendet wieder das Modul für die Überprüfung einer SQL-Bedingung. Diesmal wird kontrolliert, ob eine Übundelgsgruppe für die Vorlesung angelegt wurde. Nur wenn dies der Fall ist, kann der Schritt erledigt werden. Des weiteren wird das Modul ’module-Uebungsgruppen.inc.php’ eingebunden. Es besteht aus zwei DBTable-Submodulen. Das erste spiegelt die angelgten Übungsgruppen wieder, wobei der Tutor ausgewählt werden kann. Das zweite stellt die reservierten Räume. Aus dieser Liste können die für die Übungsgruppen vorgesehenen Räume ausgewählt werden, die dann in der oberen Tabelle als neue Übungen erscheinen.

5. “Anmeldung zur Vorlesung” ist einer der aufwändigsten Schritte. Er ruft das Modul ’module-Vorlesungsanmeldung.inc.php’ auf, das mehrere Submodule enthält. Als erstes gibt es ein Extmod-Submodul, mit dessen Hilfe eine temporäre, öffentliche Tabelle in der Datenbank angelegt wird. In diese Tabelle werden die Daten der Studenten aufgenommen, die sich zur Vorlesung angemeldet haben. Die vier folgenden Submodule sind vom Typ DBTable. Im ersten wird eine Liste der angemeldeten Studenten dargestellt, die bereits in den geschützten Bereich der DB übernommen wurden. Die nächste Tabelle stellt die in der temporären Tabelle enthaltenen Anmeldungen gruppiert nach Matrikelnummer und Universität dar. Dies ist notwendig, weil davon ausgegangen werden muss, dass Merhfachanmeldungen vorkommen. Zur jeweils obersten Zeile dieser Tabelle werden in der Tabelle des nächsten Submoduls alle zugehörigen Tupel (nicht gruppiert) dargestellt. Außerdem wird ein Eintrag aus der Relation ’student’ hinzugenommen, wenn dort einer zur aktuellen Kombination aus Matrikelnummer und Universität passt. Aus dieser Tabelle kann dann der richtige Datensatz ausgewählt werden, wodurch er in die Relation ’student_besucht_vorlesung’ eingetragen wird. Gleichzeitig werden alle zur aktullen Matrikelnummer/Universität gehörenden Tupel aus der temporären Tabelle gelöscht. Der Benutzer der Anwendung kann sich außerdem dafür entscheiden, dass die Tupel nur gelöscht werden, ohne Übernahme in den geschützten Bereich der DB. Das letzte DBTable-Submodul gibt die Relation ’student’ wieder. Auch daraus können Studenten in die Relation ’student_besucht_vorlesung’ übernommen werden. Als zweites Modul verwendet dieser Workflow-Schritt wieder ’module-SQLBedingung.inc.php’, wobei überprüft wird, ob ein Eintrag in ’student_besucht_vorlesung’ vorliegt.

6. Der Schritt “Einteilung der Übungsgruppen” verwendet das Modul ’module-Uebungseinteilung.inc.php’, das zwei Submodule und weitere interessante Details enthält. Das erste Submodul ist vom Typ DBTable und zeigt die aktuelle Belegung der Übungsgruppen, bietet aber keine Möglichkeit der Datenmanipulation. Das zweite Submodul, ebenfalls vom Typ DBTable, gibt einen Überblick, wie sich die Präferenzen Studenten

über die einzelnen Übungsgruppen verteilen. Die Tupel mit den gleichen Präferenzen aus der Relation 'student_besucht_uebung' werden dabei gruppiert. Wird die Aktion "Ändern" ausglöst, so werden die gruppierten Tupel einzeln aufgeschlüsselt und der Mitarbeiter kann bei Bedarf auf die Übungsgruppeneinteilung Einfluss nehmen. Neben den Submodulen wird im PHP-Skript eine Funktion implementiert, die eine automatische Übungsgruppeneinteilung vornimt. Allerdings wird dabei nur die erste Präferenz der Studenten berücksichtigt. Außerdem enthält das Modul-Skript nicht nur eine main-Funktion, sondern auch eine fullpage-Funktion. Damit wird im Browser eine Liste mit allen Übungsteilnehmern und den zugewiesenen Übungsgruppen ausgegeben. Diese Liste kann mit Hilfe des Browsers ausgedruckt und als Aushang verwendet werden. Das Modul für die SQL-Bedingungen prüft schließlich, dass allen Studenten, die an den Übungen teilnehmen wollen, eine Gruppe zugeteilt wurde.

7. Der Workflow-Schritt Klausur dient als Einstiegspunkt in den gleichnamigen Unterworkflow. Dazu dient das Modul 'parent_module-Klausur.inc.php'. Modul 'module-Klaururtabelle.inc.php' enthält ein DBTable-Submodul für die Verwaltung der einzelnen Klausuren. Das Modul für die SQL-Bedingungen stellt sicher, dass eine Klausur existiert.

   (a) Der erste Schritt im Klausur-Subworkflow heisst "Klausurräume festlegen" und nutzt das Modul 'module-KlausurRaum.inc.php'. Er enthält zwei DBTable-Submodule, das eine stellt die ausgewählten, das andere alle verfügbaren Räume dar. Wie üblich überprüft das SQLBedingung-Modul, ob mindestens ein Klausurraum ausgewählt wurde, bevor der Schritt erledigt werden kann.

   (b) Als nächstes folgt der Schritt "Ankündigung der Klausur". Hier wird das Modul 'module-ExternesProgramm.inc.php' eingebunden, das ein ExtProg-Submodul zum Aufruf eines externen Programms enthält. Hiermit könnte das Ausdrucken eines Aushangs oder das Freischalten einer Web-Seite initiiert werden.

   (c) Der Schritt "Anmeldung zur Klausur" ist mit dem Schritt "Anmeldung zur Vorlesung" weitestgehen identisch.

   (d) Der Schritt "Personal für die Klausuraufsicht" enthält im Modul 'module-KlausurAufsicht.inc.php' drei DBTable-Submodule. Die letzten beiden enthalten je eine Liste der Lehrstuhlmittarbeiter und der Studenten. Aus diesen Listen können Personen für die Klausuraufsicht ausgewählt werden und erscheinen dann in der Tabelle des ersten DBTable-Submoduls. Bevor der Schritt als erledigt markiert werden kann, muss mindestens eine Aufsichtsperson ausgewählt worden sein. Dies stellt eine entsprechende SQLBedingung sicher, die vom Modul 'modul-SQLBedingung.inc.php' geprüft wird.

   (e) Das Modul 'module-KlausurRaumeinteilung.inc.php' enthält zunächst eine Funktion mit der die Verteilung der Studierenden auf die Klausurräume durchgeführt werden kann. Die Verteilung erfolgt alphabetisch. Die aktuelle Einteilung wird von einem DBTable-Submodul dargestellt, wobei nach den Klausurräumen gruppiert werden. Möchte der Betreuer der Vorlesung ändern, kann er auf den Ändern-Button klicken. Die zur Gruppe gehörenden Tupel werden dann einzeln aufgelistet und die Räume können geändert werden. Darüber hinaus verwendet der Workflows-Schritt "Verteilung auf die Klausurräume" auch wieder das Modul zur Überprüfen einer SQL-Bedingung. Sie stellt diesmal sicher, dass jeder Student einen Sitzplatz hat.

   (f) Der Workflow-Schritt "Festelgen der Klausuraufgaben" nutzt die Module 'module-KlausurAufgaben.inc.php' und 'module-SQLBedingung.inc.php'. Das erste Modul enthält ein DBTable-Submodul, mit dessen Hilfe die Aufgaben verwaltet werden können. Das andere Modul kontrolliert, ob auch Aufgaben in die Datenbank eingetragen wurden.

   (g) Der Workflow-Schritt "Ausdrucken der Klausurunterlagen" ruft das Modul 'module-Drucken.inc.php' auf, das seinerseits ein ExtProg-Submodul enthält.

(h) Der Workflow-Schritt “Klausur abhalten” verwendet kein Modul, es dient nur als “Platzhalter” bis die Klausur geschrieben ist.

(i) Der Schritt “Klausur korregieren” bedient sich des Moduls 'module-KlausurKorrektur.inc.php'. Es enthält ein ExtMod-Submodul, mit dessen Hilfe ein View erzeugt und verwaltet wird. Dieser View wird für die Korrektoren freigeschaltet und dient der Eingabe der Klausurpunkte in die Datenbank. Damit auch alle Klausuraufgaben korregiert worden sind, führt das Modul 'module-SQLBedingung.inc.php' eine entsprechende Überprüfung durch.

(j) Für die “Auswertung der Klausur” ist das Modul 'module-KlausurAuswertung.inc.php' vorgesehen. Es enthält zwei Submodule vom Typ DBTable. Das erste Submodul zeigt in einer Tabelle für jeden Student die erreichten Punkte und die entsprechende Note an. Das zweite Submodul enthält eine Liste aller Studenten. Aus ihr kann ein Student in die erste Tabelle eingefügt werden, falls noch eine Klausur auftaucht die bisher nicht korregiert wurde. Darüber hinaus enthält das Modul eine Funktion, die die Noten automatisch vergibt. Der dazu notwendige Notenschlüssel kann über ein HTML-Formular eingegeben werden. Dem Erzeugen eines Aushangs dient die fullpage-Funktion im Modul. Das Modul 'module-SQLBedingung.inc.php' stellt diesmal sicher, das jeder Klausurteilnehmer mit einer Note bedacht wurde.

(k) Der Schritt “Klausur-Ende” bindet das Modul 'end_module-Klausurende.inc.php' ein. Es schließt den Subworkflow “Klausur” ab.

8. Im Hauptworkflow geht es weiter mit der “Vergabe der Scheine”. Das Modul 'module-Scheinvergabe.inc.php' enthält zwei DBTable-Submodule. Die erste listet die vergebenen Scheine auf und ermöglicht bei Bedarf Korrekturen. Die zweite Tabelle stellt die in den Klausuren erzielten Ergebnisse der einzelnen Studenten dar. Ist der Betreuer der Vorlesung der Meinung, dass das erzielte Ergebnis eines Studenten ausreicht, kann für ihn ein entsprechender Schein eingetragen werden.

9. Was noch zu tun bleibt, ist die “Fertigstellung der Scheine”. Dazu dient das Modul 'module-Scheinfertigstellung.inc.php'. Es verwendet ein ExtProg-Submodul, mit dessen Hilfe ausgewählt werden kann, welche Scheine gedruckt werden sollen. Das Drucken übernimmt ein exterenes Programm, dem die notwendigen Daten aus der Datenbank übergeben werden.

10. Der Schritt “TGI-Ende” schließt den Workflow ab.

5 Weitere Konzepte und Implementierungsdetails

5.1 Sessions

5.1.1 HTTP und Zustandsverwaltung

Das Hypertext-Transfer-Protokoll (HTTP) ist ein zustandsloses Protokoll. Dadurch kann der HTTP-Server keinen Zusammenhang zwischen einzelnen Requests eines Browsers herstellen. Jeder Request wird als eigenständig und unabhängig von vorangegangenen Requests betrachtet. Web-Anwendung bestehen jedoch in der Regel aus einer Folge von Interaktionen zwischen den Anwendern und dem Anwendungsserver. Deshalb muss der Anwendungsserver den momentanen Zustand der Anwendung sichern und einen neuen Request des Anwenders dem entsprechenden Anwendungszustand zuordnen können. Da mehrere Anwender gleichzeitig die Anwendung nutzen können, muss der Anwendungsserver in der Lage sein mehrere Anwendungszustände zu verwalten. Diese Fähigkeiten sind ein wesentliches Merkmal, was einen Anwendungsserver von einem reinen HTTP-Server unterscheidet. Der Zustand einer Anwendung kann aufgefasst werden als Belegung bestimmter Anwendungsvariablen. Bei einem Request des Anwenders müssen diesen Variablen wieder die Werte zugewiesen werden, die sie zuletzt hatten. Letztlich basiert aber auch ein Anwendungsserver auf HTTP, so dass auf ein eigentlich zustandsloses Protokoll eine Zustandsverwaltung aufgesetzt werden muss.

Die einfachste Möglichkeit besteht darin, dass der Anwender bei jedem Request seinen Zustand an den Anwendungsserver mitschickt. Dazu baut der Anwendungsserver die Variablen, die den Zustand definieren, und deren aktuelle Belegung in jede HTML-Seite ein, die er an den Browser des Anwenders sendet. Der Zustand muss in jedem Link und Formular der HTML-Seite enthalten sein, die für einen Request an den Anwendungsserver vorgesehen sind. Bei Links wird der Zustand an die URI durch ein Fragezeichen getrennt angehängt. Bei Forms ist die Verwendung eines "Hidden"-Formelements am besten geeignet. Klickt der Anwender auf einen Link oder schickt er ein Formular ab, erhält der Anwendungsserver auf diese Weise alle Werte der Zustandsvariablen zurück. Dieses Verfahren hat jedoch zwei Nachteile:

- Mit zunehmende Zahl der Zustandsvariablen wird dieses Verfahren unhandlich. Unter Umständen wird die URI mit den angehängten Variablen, so groß, dass der HTTP-Server sie nicht mehr verarbeitet.
- Das Kodieren des gesamten Zustandes in die HTML-Seiten gibt Informationen über die interne Arbeitsweise der Anwendung preis. Außerdem hat der Anwender die Möglichkeit alle Variablen abzuändern. Beides ist für die Sicherheit der Anwendung problematisch.

Auf Grund dieser Nachteile wird bei größeren Anwendungen in der Regel ein Verfahren verwendet, bei dem die Zustandsvariablen beim Anwendungsserver gespeichert werden. Anstatt alle Variablen in die HTML-Seite zu kodieren wird nur noch eine ID in die Links bzw. Formulare eingebaut. An Hand dieser ID kann der Anwendungsserver dann die zugehörigen Zustandsvariablen wiederherstellen. Darüberhinaus ist das Ablegen der ID auch in Form eines Cookies möglich, so dass dann gar keine Zustandsinformationen in der HTML-Seite enthalten sind. Wird die ID groß genug gewählt und wird sie durch einen geeigneten Zufallsalgorithmus bestimmt, ist auch das Erraten einer anderen gültigen ID praktisch nicht möglich. Da die Zustandsvariablen auf dem Anwendungsserver gespeichert werden ist die Sicherheit dieses Konzepts eng mit der Zugriffsicherheit des Rechner verbunden.

5.1.2 Das Session-Konzept von PHP

PHP bietet ab der Version 4 ein Session-Konzept, das eine Zustandsverwaltung ermöglicht. Die Zustandsvariablen und ihre Werte werden zwischen den Requests linearisiert in einer Datei gespeichert. Für jede Session wird eine solche Datei angelegt. Standardmäßig werden diese Dateien im Verzeichnis '/tmp' abgelegt. Mit dem Parameter 'session.save_path' in der PHP-Konfigurationsdatei 'php.ini' kann der Speicherort geändert werden. Man kann sogar eigene Routinen für das Speichern und Wiederherstellen von Sessions erstellen ('session.save_handler'). Damit könnten die Session-Variablen auch in einer Datenbank verwaltet werden. In der Datei 'php.ini' kann mit Hilfe der Parameter 'session.gc_maxlifetime' und 'session.gc_probability' die Gültigkeitsdauer einer Session beeinflusst werden.

Die Session-ID kann mit Hilfe eines Cookies beim Anwender gespeichert werden. Dabei wird dieses Cookie nur im Speicher erzeugt und beim Beenden des Browser gelöscht. Unterstützt der Browser keine Cookies vereinfacht PHP das Einbinden der ID in die Links bzw. Formulare durch die Stringkonstante 'SID'. Diese Konstante hat dann die Form 'session_name=session_id'. Wenn der Browser Cookies erlaubt, ist die Konstante leer.

Eine Variable muss explizit als Session-Variable deklariert werden. Dazu dient die PHP-Funktion 'session_register'. Bei einem Request werden die zur ID gehörenden Variablen wiederhergestellt, sobald die Session mit den Funktionen 'session_start' explizit oder 'session_register' implizit gestartet wird. Da die Session-ID unter Umständen über ein Cookie verwaltet wird und Cookies Bestandteil des HTTP-Headers sind, muss darauf geachtet werden, dass die Session gestartet wird, bevor das PHP-Skript eine HTML-Ausgabe erzeugt.

Auf die Variablen kann in einem PHP-Skript über deren Variablennamen zugegriffen werden, sofern in der PHP-Konfigurationsdatei 'php.ini' die Variable 'register_globals' auf 'On' steht. Ist der Wert 'Off', so können die Session-Variablen nur über das Array 'HTTP_SESSION_VARS' (indiziert mit den Variablennamen) benutzt werden. Leider ist der Schalter 'register_globals' relativ grob, da er sich nicht nur auf die Session-Variablen, sondern auf alle anderen externen Variablen ('HTTP_GET_VARS' etc.) bezieht. Für die Workflow-Anwendung muss dieser Schalter auf 'On' stehen. Wichtig ist außerdem die Reihenfolge, in der die externen Variablen zur Verfügung gestellt werden. Die Session-Variablen werden als letztes abgearbeitet, so dass eine Variable, die beispielsweise bereits aus einen HTTP-GET-Request existiert, durch den Wert einer Session-Variable überschrieben werden kann, sofern sie den selben Namen benutzen. Auf den Wert aus dem GET-Request kann dann nur noch über das Array 'HTTP_GET_VARS' zugegriffen werden.

5.1.3 Das Session-Konzept der Workflow-Anwendung

Das eben beschriebene Verhalten von PHP 4 beim Abarbeiten der externen Variablen wird von der Workflow-Anwendung ausgenutzt. Deswegen muss in der PHP-Konfigurationsdatei 'php_ini' sichergestellt sein, dass die Seesion-Variablen immer als letztes verarbeitet werden. Die Funktionen der Workflow-Anwendung legen für jede Variable, die vom Benutzer an die Anwendung geschickt werden kann, eine Session-Variable mit einem Standardwert an. Das heißt, dass der Wert jeder Variablen aus einem Link oder einem Formular (also aus 'HTTP_GET_VARS' oder 'HTTP_POST_VARS') zunächst durch den Wert der Session-Variable überschrieben wird. Erst durch den Aufruf der Funktionen 'session_update_local($name, $tiefe=0, $erweitern=0)' oder 'session_update_global($name, $tiefe=0, $erweitern=0)' wird der Wert aus dem Link oder dem Formular restauriert. Dadurch kann sichergestellt werden, dass eine Session-Variable nicht aus Versehen (oder böswillig) überschrieben wird. Jede sicherheitsrelevante Variable sollte man deshalb als Session-Variable registrieren.

Die genannten Funktionen bieten für Arrays darüber hinaus die Möglichkeit anzugeben, auf welcher Schachtelungstiefe die Ersetzung der Werte vorgenommen werden soll. Dabei bedeutet '0', dass das ganze Array ersetzt wird, '2' beispielsweise, dass die Values der 2. Dimension des Arrays ersetzt werden. Außerdem kann festgelegt werden, ob die Werte aus dem Link bzw. dem Formular das Array erweitern dürfen oder nicht. Die Funktion 'session_update_global' überschreibt den Wert der Sitzungsvariablen mit dem Wert aus dem Link oder Formular und löscht den Wert aus 'HTTP_GET_VARS' oder 'HTTP_POST_VARS'. Die Funktion 'session_update_local' überschreibt den Wert der Session-Variable nicht, sondern liefert nur eine entsprechende Rückgabe. Beide Funktionen überprüfen außerdem, ob für eine Variable eine bestimmte Funktion definiert ist. Der Name einer solchen Funktion muss mit 'session_handler_' beginnen und anschließend den Namen der Variablen enthalten. Wenn eine solche Funktion existiert, wird sie vor dem Überschreiben der Session-Variable aufgerufen, wobei ihr der Wert aus dem Link oder Formular als Referenz übergeben wird. Auf diese Weise können die Werte bearbeitet und überprüft werden oder es können andere Routinen aufgerufen werden, die bei bestimmten Variablen notwendig sind.

Die eben beschriebenen Funktionen werden von der Library 'lib-session.inc.php' bereit gestellt. Die Library enthält außerdem die Funktionen 'session_begin()' und 'session_my_register(var_liste)', die statt der PHP-Funktionen 'session_start' und 'session_register' in der Anwendung verwendet werden. Beim Aufruf von 'session_begin' wird die Konstante 'USE_SESSION' auf '1' gesetzt. Diese Konstante bestimmt, ob überhaupt das PHP-Session-Konzept verwendet werden soll. Ist die Konstante nicht gesetzt, d. h. wurde die Funktion nicht aufgerufen, gehen die anderen Funktion in 'lib-session.inc.php' und 'lib-sessionsupport.inc.php' davon aus, dass keine Sessions verwendet werden sollen und verhalten sich dementsprechend. Dies betrifft vor allem die Funktion 'session_my_register'. Ist 'USE_SESSION' gesetzt, ruft sie nur die PHP-eigene Funktion 'session_register' auf. Im anderen Fall speichert sie die Namen aller ankommenden Variablen in einem Array. Beim Aufbau einer HTML-Seite mit Hilfe der Funktionen aus der Bibliothek 'lib-sessionsupport.inc.php', wird dann versucht, die in diesem Array registrierten Variablen in die Links oder Formulare einzubauen. Dies gelingt allerdings nur, wenn die Anzahl der Variablen nicht zu groß wird. Die Funktionen, die die Variablen bei Bedarf einbauen, heißen 'session_link', 'session_link_with_image', 'session_form_head' und 'session_form' und Verhalten sich alle wie die Pendants in 'lib-html.inc.php' (siehe Abschnitt 4.6). Diese Funktionen stellen außerdem sicher, dass die Konstante SID bei Bedarf eingebaut wird und ist deshalb den entsprechenden Funktionen in 'lib-html.inc.php' vorzuziehen, sofern Session-Unterstützung erwünscht ist.

Wer Speicherplatz schonen und vielleicht bessere Performance-Werte erzielen will, kann außerdem die Funktion 'session_unregister_on_pagechange' für die in einem Modul verwendeten Session-Variablen verwenden. Dieser Funktion übergibt man eine Liste der Session-Variablen, die bei einem Seitenwechsel, sprich beim Aufruf eines neuen Workflow-Schritts, gelöscht werden sollen.

5.2 SQL-Schnittstelle

In der Datei 'lib-sql.inc.php' ist eine allgemeine Datenbank-Schnittstelle implementiert. Die hierin bereitgestellten Funktionen bieten einen generischen Zugriff auf eine SQL-Datenbank. Dafür werden allgemeine SQL-Funktionsaufrufe in die speziellen SQL-Schnittstellen-Aufrufe der jeweils verwendeten Datenbank übersetzt. Im restlichen Workflow-System werden nur diese allgemeinen SQL-Funktionen benutzt. Dies erleichtert die Umstellung auf eine andere Datenbank. Entscheidet man sich nämlich später für den Einsatz eines anderen Datenbank-Systems, so muss nur lib-sql angepasst werden.

Bisher werden von lib-sql.inc.php die Datenbank-Systeme "Mysql", "Postgres" und "Interbase" unterstützt. Für jede von diesen ist jeweils die eigentliche Umsetzung zwischen generischer und spezieller SQL-Schnittstelle in einer eigenen Datei mit Namen lib-<dbname>.inc.php implementiert , wobei <dbname> für "mysql", "psql" (für Postgres) bzw. "ibase" (für Interbase) steht. Die Funktionen in lib-sql.inc.php, die vom restlichen System benutzt werden, sind nur Wrapper, die die eigentlichen Umsetzungsfunktionen des verwendeten Datenbanksystems in 'lib-<dbname>.inc.php' aufrufen.

Die generische SQL-Schnittstelle bietet Funktionen zum Auf- und Abbau mehrerer (unterschiedlicher) Datenbank-Verbindungen, das Ausführen von SQL-Queries und den Zugriff auf die Anfrage-Ergebnisse. Dabei sollten die gestellten SQL-Anfragen so allgemein sein, wie möglich, d.h. von jedem SQL-Datenbank-System verarbeitbar und keine Implementierung-spezifischen Details enthalten.

Da die SQL-Systeme aber vor allem bei View- und Rechte-Verwaltung unterschiedlich zu handhaben sind (mysql kennt z.B. von sich aus gar keine Views, sie werden im Workflow-System mit temporären Tabellen nachgeahmt), existieren für diese Aufgaben pro unterstütztem Datenbank-System eigene Funktionen, die über die allgemeine Schnittstelle ansprechbar sind.

Auch die Verarbeitung von Anfragen, die mit einen UNION-Operator gebildet werden, ist sehr unterschiedlich. "Mysql " unterstützt diesen Vereinigungs-Operator gar nicht. Aber auch "Postgres" und "Interbase" haben hierbei Typ-Konvertierungs-Schwierigkeiten, falls leicht unterschiedliche SQL-Typen (z.B. 2 verschiedene Arten von Strings) verwendet werden. Bei

Postgres wäre dies nur durch implementierungs-abhängiges Type-Casting zu lösen. Da jedoch im Workflow-System oft SQL-Anfragen zu stellen sind, die sich aus mehreren zu vereinigenden Teilanfragen zusammensetzen, bietet lib-sql bereits die Möglichkeit eine generische Anfrage zu stellen, die aus Teilanfragen besteht. Dabei wird die Anfrage in die einzelne Teilanfragen zerlegt und diese einzeln auf die speziellen Datenbank-Systeme umgesetzt. Typ-Überprüfungen werden hierbei jedoch von lib-sql nicht durchgeführt. Die Einzel-Ergebnisse werden einfach hintereinandergehängt, d.h. Duplikate werden nicht entfernt und Sortierungen auf dem Gesamtergebnis sind nicht möglich. Dies entspricht der Wirkung des SQL-Operators "UNION ALL". Dieser wird somit aber einheitlich von lib-sql für alle Systeme (auch mysql) zur Verfügung gestellt. Das Zerlegen in Teilanfragen bzw. das Zusammensetzen des Ergebnisses der Anfragen ist in der Datei 'lib-sqlunion.inc.php' realisiert. Wo es nötig ist, werden daher diese Funktionen (und nicht die tatsächlichen spezifischen Funktionen) von den Wrapper-Funktionen in 'lib-sql.inc.php' aufgerufen und diese rufen ihrerseits die tatsächlich Datenbank-spezifische Funktion auf.

Bei SQL-Anfragen ist auf Schreibweise von Relations- und Attributsnamen zu achten. Da es Datenbanken, wie z.B. "mysql" gibt, die bei diesen Namen Groß- und Kleinschreibung unterscheiden, wurde Kleinschreibung als einheitliche Schreibweise für solche Namen gewählt.

Im folgenden werden die Funktionen der SQL-Schnittstelle genauer beschrieben.

Die Namen der generischen Funktionen beginnen immer mit 'db_', die Namen für Funktionen der "UNION"-Behandlung immer mit 'db_union'. Die Namen der speziellen Datenbank-System-spezifischen Funktionen beginnen zur Zeit immer mit 'my_<dbtype>', wobei <dbtype> durch "mysql" , "pg" (für "Postgres") oder "ibase" (für "Interbase") zu ersetzen ist. Das 'my' dient hierbei dazu, diese Funkionen von den jeweiligen php-internen <dbtype>-Schnittstellen-Funktionen (die i.d.R. von diesen 'my-<dbtype>' sogar verwendet werden) zu unterscheiden.

5.2.1 Funktionen für die Verwaltung von Datenbank-Verbindungen

db_connection_register($database_array, $name="default", $connect_immediatelly=0): Mit der Funktion 'db_connection_register' kann eine neue Datenbank-Verbindung angemeldet werden. Das assoziative array $database_array spezifiziert dabei die Verbindung. Am wichtigsten ist hierbei $database_array[type], der das zu verwendende Datenbank-system für die neue Verbindung angibt. Als Werte sind im Moment "mysql", "postgres" oder "interbase" möglich. Der Rest des Arrays enthält Parameter für das jeweils verwendete Datenbank-System:

Für "mysql" sind dies die Parameter "host", "username", "password", "database",
für "Postgres" "host", "port", "user", "password", "database" und
für "Interbase" "host", "username", "password", "database".

Um die Verbindung später für Anfragen zu benutzen wird sie über $name noch mit einem sie eindeutig identifizierenden, globalen Namen versehen, der als Standardwert "default" hat.

Außerdem wird, falls $connect_immediatelly den Wert 1 hat, die Verbindung sofort beim Aufruf von 'db_connection_register' aufgebaut. Sonst wird die Verbindung erst bei Bedarf, d.h. bei der ersten SQL-Anfrage aufgebaut. Damit werden unnötige Datenbankverbindungen, die später gar nicht benutzt werden, unterlassen.

Der Rückgabewert bei Fehler ist 'false', sonst 'true'.

Verwendet werden im aktuellen Workflow-System zur Zeit 2 Datenbank-Verbindungen:

Eine spezielle mit Namen "control" für die Verwaltung von Views und Rechten, da hier ein Datenbank-Benutzer mit speziellen Zugriffs-rechten nötig ist, und die Standardverbindung "default" für sonstige Aufgaben.

Schlägt die Registrierung fehl (z.B. weil eine Verbindung mit diesem Namen schon angemeldet ist), so liefert 'db_connection_register' 'false' als Rückgabewert, sonst 'true'.

Intern in lib-sql wird die Funktion 'db_connection_prepare' verwendet, um eine Verbindung tatsächlich aufzubauen, falls noch nicht geschehen. Sie wird i.d.R benutzt, um sicherzustellen, dass eine Verbindung aufgebaut ist, bevor sie für Anfragen benutzt wird.

Den eigentlichen Aufbau einer Verbindung übernimmt die Funktion 'db_connect', die die Datenbank-spezifische (abhängig von $database_array[type]) connect-Funktion aufruft. Diese spezifische Funktion erhält als Parameter das die neue Verbindung spezifizierende $database-Array und gibt einen Datenbankverbindungs-Handle (abhängig vom Datenbank-Typ) oder 'false' bei Fehler zurück. Die momentan implementierten Datenbank-System-spezifischen Funktionen sind 'my_mysql_connect' (für "mysql"), 'my_pg_connect' (für "Postgres") und 'my_ibase_connect' (für "Interbase").

db_connection_unregister($name="default"): Abgemeldet wird eine Datenbank-Verbindung mit 'db_connection_unregister'. Als Parameter ist nur der Name der Verbindung nötig, der bei "default" auch entfallen kann.

Bei erfolgreicher Abmeldung wird 'true' zurückgegeben, sonst 'false' (z.B. falls Verbindung gar nicht vorhanden).

Außerdem wird die Datenbankverbindung, falls sie tatsächlich aufgebaut (d.h. tatsächlich benutzt) wurde, über die Funktion 'db_close' geschlossen. 'db_close' hat als Parameter das $database-Array (in dem von 'db_connection_prepare' unter dem Key "db_handle" das tatsächliche System-spezifische Datenbankverbindungs-Handle gespeichert ist) und liefert als Rückgabe 'true' bzw. 'false' (bei Fehler). Sie benutzt hierfür die System-spezifische close-Funktion. Momentan tragen alle DB-spezifischen close-Funktionen stets den Namen 'my_<dbtype>_close'.

5.2.2 Funktionen für das Ausführen von SQL-Anfragen

db_query($querystring, $connection_name="default"): Um SQL-Anfragen auszuführen wird die Funktion 'db_query' verwendet. Als Parameter hat sie den SQL-Query-String und den Namen der Datenbank-Verbindung. Für die Standardverbindung "default" kann der Name entfallen. Die Anfrage sollte so allgemein sein, dass jedes verwendbare Datenbanksystem ("mysql", "Postgres" und "Interbase") sie verstehen und auswerten kann.

"UNION" im Sinne von "UNION ALL" wird mit Hilfe der Funktion 'db_union_query' von 'lib-sqlunion.inc.php' unabhängig vom zugrunde liegenden Datenbank-System unterstützt. Dazu wird von 'db_query' 'db_union_query' aufgerufen, die den Anfragestring zerlegt und die Einzelanfragen an das darunter liegende Datenbank-System weitergibt.

Bei Fehler wird von 'db_query' 'false' als Rückgabe geliefert, sonst der Result-Handler (der ungleich 'false' ist). Über diesen kann dann mit den weiter unten beschrieben Funktionen auf das Anfrage-Ergebnis zugegriffen werden.

Die zur Zeit implementierten Datenbank-spezifischen Funktionen sind 'my_mysql_query' (für "mysql"), 'my_pg_query' (für "Postgres") und 'my_ibase_query' (für "Interbase"). Sie werden von 'db_union_query' aus aufgerufen.

db_errormessage($connection_name="default"): Mit 'db_errormessage' kann die letzte Fehlermeldung innerhalb einer Datenbank-Verbindung abgefragt werden. Die Fehlermeldung ist nur ein String. Er ist abhängig vom verwendeten Datenbank-System.

Die spezielle Funktion für das einzelne Datenbank-System heißt 'my_<dbtype>_errormessage'.

db_commit($connection_name="default"): Speziell für Datenbank-Systeme, die eine Transaktionsverwaltung beim SQL-Zugriff explizit vorsehen, d.h. für Interbase, wurde 'db_commit' eingeführt um, alle bisherigen Änderungen innerhalb einer Datenbank-Verbindung zu bestätigen. Im Falle von "Postgres" und "mysql" hat diese Funktion keine Wirkung. Sie sollte aber für die korrekte Zusammenarbeit mit Datenbank-Schnittstellen, wie die von Interbase, stets benutzt werden.

5.2.3 Funkionen für den Zugriff auf Metadaten eines Anfrage-Ergebnisses

Folgende Funktionen dienen dem Zugriff auf Metadaten, wie z.B. Zahl und Namen der Attribute, eines SQl-Anfrage-Ergebnisses:

db_numfields($result): Die Zahl der Spalten, d.h. die Zahl der Attributswerte einer Zeile, bei einer SQL-Anfrage ist über die Funktion 'db_numfields' verfügbar. Als Parameter ist nur der Result-Handler der Anfrage erforderlich.

Tritt kein Fehler auf, so wird die Anzahl der Ergebnis-Spalten zurückgegeben, sonst 'false'.

Derzeit wird stets 'db_union_numfields' mit dem <dbtype> aus $result aufgerufen. Bei mit "UNION" zusammengesetzten Anfragen wird in der Funktion 'db_union_numfields' aus 'lib-sqlunion.inc.php' die Anzahl der Spalten des Ergebnisses der ersten Teilanfrage zurückgegeben und eine Warnung erzeugt, falls diese nicht mit den Spaltenzahlen der übrigen Teilergebnissen übereinstimmt. Die von 'db_union_numfields' aufgerufene, tatsächlich Datenbank-spezifische Funktion heißt 'my_<dbtype>_numfields'.

db_fieldname($result, $num): Mit 'db_fieldname' kann man den Attributsnamen einer Spalte eines Ergebnisses bestimmen. Als Parameter dienen der Result-Handler der SQL-Anfrage und die Attributsnummer $num von 0 an gezählt.

Bei Fehler wird 'false' zurückgegeben, sonst der Attributsname. Der Attributsname wird nach der oben genannten Konvention "klein-geschrieben" zurückgegeben, selbst wenn das darunter liegende Datenbank-System ihn in anderer Schreibweise liefert.

Derzeit wird stets die Funktion 'db_union_fieldname' benutzt. Bei mit "UNION" zusammengesetzten Anfragen werden die Attributsnamen der ersten Teilanfrage benutzt (siehe 'db_union_fieldname' aus 'lib-sqlunion.inc.php'). Die tatsächliche Datenbank-abhängige dabei Funktion ist 'my_<dbtype>_fieldname'.

db_numrows($result): Bei Datenbanken, die dies unterstützen (d.h. "Postgres" und "mysql") kann mit dieser Funktion die Anzahl der Zeilen, d.h. der Tupel,des Ergebnisses einer SQL-Anfrage als Rückgabe geliefert werden. Da jedoch "Interbase" dies nicht unterstützt, ist diese Funktion nicht universell einsetzbar und wird im aktuellen System nicht benutzt.

Derzeit wird stets die Funktion 'db_union_numrows' mit <dbtype> als Parameter benutzt. Bei zusammengesetzten "UNION"-Anfragen werden die Summe der Anzahlen aller Teilanfragen benutzt. Die tatsächliche Datenbank-abhängige Funktion heißt stets 'my_<dbtype>_numrows'.

db_numrows_withseektoend($result): Um auch bei Datenbanken wie "Interbase" überhaupt auf die Anzahl der Zeilen eines SQl-Anfrage-Ergebnisses zuzugreifen wurde 'db_numrows_withseektoend' eingeführt. Wenn möglich sollte nur diese Funktion und nicht 'db_numrows' verwendet werden. Sie hat wie 'db_numrows' den Result-Handler der Anfrage als Parameter und gibt die Anzahl der Zeilen zurück.

Allerdings ist i.a. das gesamte Anfrage-Ergebnis nach Aufruf dieser Funktion nicht mehr abrufbar. Bei "Interbase" wird sie nämlich durch Durchlauf durch das gesamte Ergebnis realisiert und "Interbase" kann nur einmal auf eine Ergebniszeile zugreifen. Die anderen Datenbanken simulieren dies durch Setzen des Result-Handlers auf das Ende des Ergebnisses.

Diese Funktion sollte also erst verwendet werden , wenn alle benötigten Anfrage-Zeilen ausgelesen wurden, z.B. wenn man nur einen Ausschnitt einer SQL-Anfrage auf der aktuellen Seite darstellt und aber zusätzlich die Gesamt-Zeilenzahl anzeigen möchte.

Bisher wird immer 'db_union_numrows_withseektoend' aufgerufen, um mit "UNION" zusammengesetzte Anfragen zu behandeln. Der Name hiervon aufgerufenen, tatsächlich Datenbank-abhängigen Funktion ist derzeit immer 'my_<dbtype>_numrows_withseektoend'.

5.2.4 Funktionen für den Zugriff auf Daten des Ergebnisses einer SQL-Anfrage

Um alle verwendbaren Datenbanksystem-Schnittstellen zu unterstützen, erfolgt der Zugriff auf Anfrage-Ergebnisse zeilenweise fortlaufend, angefangen bei der 1. Zeile. "Postgres" und "mysql" könnten besseren Zugriff auf das Ergebnis bieten, d.h. z.B. Teile des Ergebnisses mehrmals durchlaufen, indem zu bereits gelesenen Zeilen zurückgesprungen wird. Da aber "Interbase" dies nicht unterstützt, wurde nur diese einfache Zugriffsart gewählt.

(Funktionen für den (evtl. mehrmaligen) Zugriff auf eine bestimmte Ergebnis-Zeile wurden zwar auch implementiert, sie sind aber nur beim Einsatz von "Postgres" oder "mysql" verwendbar. Das gesamte System benutzt daher bisher nur die einfache Zugriffs-Variante).

db_fetchnext_row($result): Mit dieser Funktion kann man die die nächste Zeile eines Anfrage-Ergebnisses als normale (nummerierte) Liste der Attributswerte abfragen. Als Parameter wird nur der Result-Handler der Anfrage benötigt. Die Rückgabe ist eine normale normale Liste der Attributswerte, d.h. ein Array indiziert mit den Nummern der Attribute, beginnend bei 0. Gibt es jedoch keine weitere Zeile mehr oder tritt ein Fehler auf, so wird statt dessen 'false' zurückgegeben (in php es jedes Array ungleich 'false').

Im Ergebnis wird durch diese Funktion eine Zeile weiter gesprungen.

'db_fetchnext_row' ruft i.a. die Datenbank-spezifische fetchnext-Funktion auf. In der aktuellen Implementierung wird zur Unterstützung des "UNION ALL"-Operators aber die Funktion 'db_union_fetchnext' als Wrapper verwendet. Diese wird mit dem Datenbank-typ <dbtype> und dem UNION-Result-Handler (beide in $result enthalten) und dem festen Funktionstyp "row" aufgerufen. Diese ermittelt dann mit der tatsächlich DB-spezifischen Funktion 'my_<dbtype>_fetchnext', die u.a. auch den Funktionstyp "row" als Parameter erhält, die nächste Zeile.

db_fetchnext_array($result): Ähnlich wie 'db_fetchnext_row' liefert diese Funktion die nächste Zeile im Ergebnis, allerdings nicht in Form einer normalen Liste, sondern als assoziatives Array. Dabei können die Attributswerte nicht nur über die Attributsnummer, sondern auch über den Attributsnamen abgefragt werden.:

z.B. kann bei der Anfrage "SELECT typ, bemerkung, wert FROM tabelle" mit $zeile[1] oder mit $zeile[bemerkung] auf den Wert des Attributs "bemerkung" zugegriffen werden, falls $zeile die Rückgabe von 'db_fetchnext_array' enthält.

Die als Schlüssel dienenden Attributsnamen sind alle lower-case.

Einziger Parameter der Funktion ist wie bei 'db_fetchnext_row' der Result-Handler. Die Rückgabe ist das oben beschriebene assoziative Array der Attributswerte der nächsten Zeile. Beim Erreichen des Ende des Ergebnisses oder bei Fehler wird statt dessen 'false' zurückgegeben. Auch diese Funktion springt im Ergebnis eine Zeile weiter.

Wie bei 'db_fetchnext_row' wird in der derzeitigen Implementierung 'db_union_fetchnext' aufgerufen, allerdings mit Funktionstyp "array" statt "row". Diese benutzt dann die tatsächlich DB-spezifische Funktion 'my_<dbtype>_fetchnext' u.a. mit dem Funktionstyp "array" als Parameter auf.

Nur, wenn "mysql" oder "Postgres", und nicht "Interbase" als Datenbank-System verwendet wird, sind die beiden folgenden Funktionen zum direkten Zugriff auf eine bestimmte Ergebnis-Zeile nutzbar. Zu erwähnen ist, das die Benutzung dieser Funktionen die Zeilen-Position im Ergebnis für die eben beschriebenen 'db_fetchnext-Funktionen' nicht ändert:

db_fetch_row($result, $row): Diese Funktion gibt eine per absoluter Zeilennummer spezifizierte Zeile des Ergebnisses als normale Liste der Attributswerte zurück. Als Parameter sind der Result-Handler und zusätzlich die Zeilennummer (angefangen bei 0) zu übergeben. Die Rückgabe ist 'false' bei Fehler (z.b. falls die Zeile nicht existiert), sonst die angeforderte Ergebniszeile als Liste (d.h. i der gleichen Form wie die nächste Zeile des Ergebnisses von 'db_fetchnext_row' zurückgegeben wird).

Ähnlich den fetchnext-Funktionen wird derzeit die Funktion 'db_union_fetch' u.a. mit dem Funktionstyp "row" als Parameter aufgerufen. Diese nutzt dann die tatsächlich DB-spezifischen Funktion 'my_<dbtype>_fetch'.

db_fetch_array($result, $row): Ähnlich wie 'db_fetch_row' liefert diese Funktion eine bestimmte Ergebnis-Zeile zurück, allerdings nicht als normale Liste, sondern als assoziatives Array der Attributsnamen und Attributsnummern (Form wie bei ’db_fetchnext_array’). Parameter sind die gleichen wie bei ’db_fetch_row’.

Auch hier wird derzeit ’db_union_fetch’ aufgerufen (mit Funktionstyp “array” als Parameter) aufgerufen, die ihrerseits die tatsächlich DB-spezifische Funktion ’my_<dbtype>_fetch’ aufruft.

db_seek ($result, $row): Um die für die ’db_fetchnext’-Funktionen verwendete Zeilenposition im Ergebnis zu ändern, kann ’db_seek’ verwendet werden. Als Parameter sind der Result-Handler und eine absolute Zeilennummer (von 0 an gezählt) zu übergeben. Bei Erfolg wird ’true’ zurückgegeben. Die aktuelle Zeilenposition ist dann vor der spezifizierten Zeile. D.h. beim nächsten Aufruf von ’db_fetchnext_row’ bzw. ’db_fetchnext_row’ wird dann die ($row+1)-te Zeile gelesen. Bei Fehler, z.B. wenn die angegebene Zeile nicht zu erreichen war, wird ’false’ zurückgegeben. Trotzdem wird immer versucht zu einer Zeile mit möglichst hoher Nummer, aber nie höher als mit $row angegeben, zu springen.

Verwendet wird hierfür die Funktion ’db_union_seek’, um mit “UNION ALL” zusammengesetzte Anfragen zu behandeln. Diese ruft die tatsächlich Datenbank-spezifische Funktion ’my_<dbtype>_seek’ auf, um auf den Teilergebnissen den Sprung durchzuführen. Diese müssen dafür (intern) als Rückgabe immer die erreichte Position innerhalb ihres Teilergebnisses als Rückgabe an ’db_union_seek’ liefern.

Die folgenden Funktionen rufen selbst (zumindest nicht direkt) keine Datenbank -spezifischen bzw. UNION-Query-behandelnde Funktionen auf. Sie benutzen nur die obigen Zugriffsfunktionen, um dem Benutzer den Zugriff auf das gesamte Ergebnis einer SQL-Anfrage in einem Funktionsaufruf zu ermöglichen:

db_fetchall_row ($result): Diese Funktion liefert die Liste aller noch übrigen Zeilen des Ergebnisses (jeweils selbst als Liste der Attributwerte) zurück, d.h. sie ruft solange wie möglich ’db_fetchnext_row’ auf. Sie benötigt nur den Result-Handler als Parameter.

db_fetchall_array ($result): ’db_fetchall_array’ arbeitet wie db-fetchall_row, benutzt aber ’db_fetchnext_array’ und liefert damit die Liste aller noch verbleibenden Ergebnis-Zeilen, wobei jede Zeile ein assoziatives Array der Attributsnamen und Attributsnummern ist, und keine normale Liste.

db_fetchall_array_list($result, $attribute): Diese Funktion liefert eine Liste der Werte eines per Nummer oder Namen gegebenen Attributs von allen verbleibenden Zeilen eines Ergebnisses zurück. Sie benutzt hierfür ’db_fetchnext_array’.

5.2.5 Funktionen für Informationen über Eigenschaften von Relationen

Da der Zugriff auf Metadaten von Relationen sehr vom verwendeten Datenbank-System abhängig ist, existieren generische Funktionen um auf einige wichtige Relations-Metadaten zuzugreifen. Die Besonderheiten der jeweiligen Datenbank werden dabei vor dem Benutzer verborgen:

db_primkey($table, $connection_name=”default”, $usecache=1): Mit dieser Funktion sind die Primary-Key-Attribute einer gegebenen Relation bestimmbar. Als Argumente werden der Relationsname ($table), die zu verwendende Datenbankverbindung, sowie optional ein Flag $usecache zu übergeben. Die Rückgabe ist eine normale Liste aller Attributsnamen, die Teil des Primary-Key der Relation $table sind. Tritt jedoch ein Fehler auf (z.B. ungültiges $connection_name), so wird ’false’ zurückgegeben.

Falls das Flag $usecache gleich 1, so wird ein Cache für bereits berechnete Ergebnisse innerhalb dieses Programmablaufes verwendet.

Die Datenbank-spezifischen Funktionen haben bisher alle den Namen ’my_<dbtype>_primkey’.

db_attributes($table, $connection_name=”default”, $usecache=1): ’db_attributes’ ist ähnlich zu ’db_primkey’. Sie hat die gleichen Argumente. Allerdings gibt seine normale Liste aller Attribute (nicht nur der Primary-Key-Attribute) der Relation $table zurück. Auch hier kann ein Cache verwendet werden.

Der Name der spezifischen Funktionen ist zur Zeit immer ’my_<dbtype>_attributes’.

5.2.6 sonstige Funkionen: View- und Rechteverwaltung, etc.

Auch die Verwaltung von Benutzerrechten, insbesondere in Verbindung mit Views ist bei verschiedenen Datenbanken sehr unterschiedlich. Daher gibt es auch hier generische Funktionen, um dem Benutzer die damit verbundenen Aufgabe n zu erleichtern:

db_grant($objekt, $user, $rechte=””, $connection_name=”default”): Um einem Benutzer Zugriffsrechte auf eine Relation zu geben dient ’db_grant’. Als Parameter sind der Relationsname $objekt, der Datenbank-Benutzername $user, die SQL-Rechte-Beschreibung $rechte (z.b. ...???), sowie die zu verwendende Datenbankverbindung. Bei Auftreten eines Fehlers wird ’false’ zurückgegeben, sonst ’true’.

Die Datenbank-spezifische Funktion heißt bei der bisherigen Implementierung stets ’my_<dbtype>_grant’.

db_grant_on_view($objekt, $user, $rechte=””, $connection_name=”default”) : Um für einen Datenbank-Benutzer bestimmte Rechte an einem View zu vergeben sind evtl. zusätzlich Datenbank-spezifische Aktionen nötig. Daher sollte zum Vergeben von SQL-Rechten für Views ’db_grant_on_view’ und nicht ’db_grant’ benutzt werden. $objekt ist dabei der Name des Views. Ansonsten sind Parameter und Rückgabe die gleichen wie bei ’db_grant’.

Die Datenbank-spezifische Funktion für “Interbase” heißt ’my_ibase_grant_on_view’, bei “mysql” bzw. “Postgres” wird einfach ’my_<dbtype>_grant’ verwendet, da keine View-spezifischen Aktionen bei der Vergabe von Rechten notwendig sind.

db_revoke($objekt, $user, $rechte=””, $connection_name=”default”): ’db_revoke’ bewirkt das Gegenteil von ’db_grant’, d.h. entzieht einem Datenbank-Benutzer bestimmte Rechte an einer gegebenen Relation. Parameter und Rückgabe haben die gleiche Bedeutung wie bei ’db_grant’.

Der Name der Datenbank-spezifischen Funktion ist immer ’my_<dbtype>_revoke’.

db_revoke_on_view($objekt, $user, $rechte=””, $connection_name=”default”): ’db_revoke_on_view’ verhält sich ähnlich wie ’db_revoke’, entzieht aber die Rechte an einem View und nicht einer normalen Relation. Die Parameter und der Rückgabewert haben ansonsten die gleiche Bedeutung wie bei ’db_revoke’ bzw. ’db_grant_on_view’.

Bei “Interbase” ist die Datenbank-spezifische Funktion ’my_ibase_revoke_on_view’, bei “mysql” und “Postgres” wird einfach ’my_<dbtype>_revoke’ benutzt.

5.3 Views und temporäre Tabellen

In Datenbanksystemen unterscheidet man die folgenden drei Ebenen:

1. In der physischen Ebene ist festgelegt, wie die Daten physisch gespeichert sind und welche Zugriffsrechte bestehen. Dabei spielen der Datentyp und die Länge eines Attributs, der Aufbau der gespeicherten Datensätze, und Indexorganisation eine wichtige Rolle. Die Effizienz der physischen Ebene bestimmt maßgeblich das Leistungsverhalten des DBMS.

2. In der logischen Ebene wird in einem Datenbankschema festgelegt, welche Daten gespeichert sind. Dadurch werden die Objekt- und Beziehungstypen sowie der Wertevorart beschrieben. Die Daten und Beziehungen werden dabei unabhängig von der tatsächlichen Speicherung betrachtet.

3. Die Sichten stellen Teilmengen der Informationen bereit. Sie sind auf die Bedürfnisse der jeweiligen Benutzer bzw. Benutzergruppen abgestimmt. Insbesondere soll kein Benutzer Daten sehen, die er nicht sehen will (Übersichtlichkeit) oder nicht sehen darf (Datenschutz). [Skript Kriegel DBS I, Kemper/Eickler Datenbanksysteme]

Auch für die Datenbank der Workflow-Anwendung können verschiedene Benutzer bzw. Benutzergruppen identifiziert werden. Neben den Mitarbeitern des Lehrstuhls sollen auch Studenten und Tutoren auf die Datenbank zugreifen dürfen. Während den Mitarbeitern eine uneingeschränkte Sicht auf die Daten gewährt werden kann, sollen die anderen Benutzergruppen nur eine notwendige Teilmenge der Daten sehen. Für einen Tutor könnte man beispielsweise einen View auf einzelne Datentupel der Relation 'student_erhaelt_punkte' einrichten. Dies sollten nur Datentupel sein, die sich auf eine Aufgabe eines Übungsblatts oder einer Klausur beziehen, die der Tutor zu korrigieren hat. Schlecht wäre es, wenn der Tutor auch auf Klausuraufgaben Zugriff hätte, die er selber geschrieben hat.

Ein View wird in SQL mit Hilfe des Befehls 'CREATE VIEW <viewname> AS' erzeugt. Dabei ist an das Schlüsselwort 'AS' eine SELECT-Anfrage anzuhängen, die die Daten des Views definiert. Neben der Erzeugung des Views muss die Benutzung desselben für einen Benutzer autorisiert werden. Dazu dient das GRANT-Kommando. Mit diesem Kommando wird außerdem festgelegt, welche Zugriffsmöglichkeiten dem Benutzer gewährt werden sollen. Die üblichen Standardprivilegien sind SELECT, INSERT, UPDATE, DELETE und REFERENCES. Mit dem REFERENCES-Privileg erhält man die Möglichkeit Fremdschlüssel zu referenzieren.

Die getesteten DBMS MySQL, PostgreSQL und Interbase unterstützen Views nur teilweise oder gar nicht:

- MySQL kennt das Konzept der Views in der vorliegenden Version nicht. In der Dokumentation zu MySQL wird als gleichwertiges Konzept die Verwendung temporärer Tabellen vorgeschlagen. Im Gegensatz zu Views stellen temporäre Tabellen jedoch keine Sicht auf die DB dar, sondern sind eigenständige Objekte, die zusätzlich zu den Objekten und Beziehungen des eigentlichen Datenbankschemas existieren. Die Daten in den temporären Tabellen sind nur immer so aktuell, wie sie zum Zeitpunkt des Erzeugens der temporären Tabelle oder des letzten Abgleichs waren. Werden Datenmanipulationen vorgenommen, müssen diese manuell mit den übrigen Daten abgeglichen werden. Die Datenbank ist so zwischen zwei Abgleichungen nicht konsistent, unter Umständen können Konflikte entstehen. Außerdem bedeutet das manuelle Abgleichen, dass der damit verbundene Aufwand nicht vom DBMS, sondern von einem externen Skript erbracht werden muss.

- PostgreSQL unterstützt in der betrachteten Version Read-Only-Views. Das heißt, dass mit den Views keine Datenmanipulationen möglich sind. Ein Update hätte beispielsweise keine Auswirkungen auf die hinter dem View stehenden realen Datenbank-Objekte. Allerdings bietet PostgreSQL die Möglichkeit mit Hilfe so genannter "Rules" diese Beschränkung zu umgehen. Vom Prinzip sind Rules spezielle Trigger, die bei einer Datenmanipulation auf einem View aktiv werden und die entsprechende Datenmanipulationen auf den realen DB-Objekten durchführen. Diese Rules müssen allerdings vom Anwendungsentwickler einzeln eingerichtet werden.

- Am weitesten geht die Unterstützung von Views bei Interbase. Interbase richtet auch Write-Through-Views ein, sofern u. a. folgende Bedingungen erfüllt sind. Der View stellt eine Teilmenge einer einzigen Relation dar. Nicht eingeschlossene Attribute müssen NULL-Werte zulassen. Werden bei der Definition des Views hingegen Joins, Aggregationen oder Subqueries verwendet kann Interbase auch nur einen Read-Only-View einrichten. Da man in einer einigermaßen gut normalisierten Datenbank ohne Joins fast nicht auskommt, ist folglich die Write-Through-Unterstützung in der Praxis nicht benutzbar. Was bleibt ist wie bei PostgreSQL durch Trigger die Datenmanipulationen auf den realen DB-Objekten nachzuführen. Positiv ist anzumerken, dass Interbase bei Write-Through-Views auch die Option WITH CHECK unterstützt. Dadurch kann folgendes sichergestellt werden: Ein Datentupel kann

Parameter	Beschreibung
id	String, mandatory. Die Views müssen für jeden Workflow eindeutige IDs erhalten. Die View-Namen selber werden von der Anwendung generiert.
query_array	Array, mandatory. Der Aufbau entspricht der Beschreibung in Abschnitt 4.3.4.
tabelle	String, mandatory. Auf diese Tabelle sollen die Datenmanipulationen durchgeschrieben werden.
user	String. Durch Komma getrennte Liste von Benutzernamen. (Gruppen werden durch das Schlüsselwort GROUP gekennzeichnet.) Für diese Benutzer werden standardmäßig Zugriffsrechte auf den View vergeben
rechte	String oder Array. Durch Komma getrennte Liste der Zugriffsrechte. Bei der Verwendung der Array-Versionen können Benutzernamen als Keys verwendet werden. Die Zugriffsrechte im Value werden dann diesem Benutzer gewährt. Bei der Verwendung der String-Version, werden die Rechte den Benutzern im Parameter 'user' zugewiesen.
view_name	String. Der Viewname wird normalerweise von der Anwendung generiert. Wem dies nicht gefällt kann hier den Viewnamen vorgeben. Dabei ist allerdings darauf zu achten, dass der Name noch nicht vergeben ist.
req	Array. Attribut-Werte-Paare. Key: Attributname, Value: Wert. Diese Werte werden bei der Erzeugung neuer Tupel durch den View grundsätzlich verwendet.

Tabelle 20: Parameter der Funktion 'view_create'

nicht so abgeändert werden, dass es nicht mehr zur Definition des Views passt. Außerdem kann kein neues Datentupel erzeugt werden, das nicht zum View passt. Negativ ist noch anzumerken, dass Interbase Probleme bekommt, wenn bei der Definition des Views Alias-Namen verwendet werden.

Wir haben versucht mit den Funktionen in der Library 'lib-view-control.inc.php' die beschriebenen Probleme der DBMS mit Views abzumildern. Mit Hilfe der Funktion 'view_create' können Write-Through-Views erzeugt werden. Die Parameter der Funktion sind in Tabelle 20 knapp dargestellt (genaueres und Informationen zu den anderen Funktionen siehe Quelltext). Bei Interbase und PostgreSQL werden dazu automatisch Trigger bzw. Rules generiert, im Falle MySQL wird versucht das View-Konzept mit temporären Tabellen nachzubilden. Bei der Definition des Views können auch Joins verwendet werden. Allerdings wird das Durchschreiben von Datenmanipulationen durch den View hindurch nur auf eine DB-Relation ermöglicht. Der Name dieser Relation muss der Funktion 'view_create' als Argument übergeben werden. Auch in unserem Konzept werden keine Aggregationen bzw. Gruppierungen unterstützt.

Die Rules bzw. Trigger, die bei Postgres und Interbase für Datenmanipulationen auf Views notwendig sind, werden in 'lib-view-control.inc.php' automatisch angelegt. Dabei richtet sich die Library nach den Rechten, die für einen View gelten sollen. Wird beispielsweise einem Benutzer das Update-Recht auf einen View erteilt, erzeugt die Library einen geeigneten Update-Trigger. Gleiches gilt für Insert und Delete. Eine wichtige Rolle spielen dabei die Primärschlüssel-Attribute der Relation, auf die die Datenmanipulationen durchgeschrieben werden. Diese können über den

View nicht mit einem Update geändert werden, da sonst der Zusammenhang zwischen dieser Relation und dem View nicht mehr einfach herzustellen ist. Bei Insert-Operationen berücksichtigt der entsprechende Trigger außerdem ein Attribut-Werte-Array, das der Funktion 'view_create' übergeben werden kann. Damit können Werte für bestimmte Attribute fest vorgegeben werden, die beim Erzeugen eines neuen Tupels auf jeden Fall verwendet werden.

Da MySQL keine Views kennt, haben wir versucht mit temporären Tabellen ein Ersatz zu schaffen, den wir Pseudoviews genannt haben. Dabei kann die gleiche Schnittstelle für die Pseudoviews verwendet werden, wie bei den echten Views. D. h. die Funktionen 'view_create', 'view_drop' etc. sind auch bei MySQL verwendbar. Wer ganz bewusst Pseudoviews erzeugen will, z. B. auch bei der Verwendung des DBMS PostgreSQL, kann die Funktionen 'pseudoview_create' etc. aufrufen. Der Nachteil der Pseudoviews ist, dass kein automatischer und sofortiger Abgleich mit der Relation stattfindet, auf die die Datenmanipulationen durchgeschrieben werden sollen. Dies muss "manuell" mit Hilfe der Funktion 'pseudoview_update' ausgelöst werden. Diese Funktion wird auf jeden Fall vor dem Löschen des Pseudoviews aufgerufen. Existiert ein Pseudoview längere Zeit, ist es unter Umständen sinnvoll, einen Cron-Job einzurichten, der die Updates in bestimmten Zeitintervallen erledigt.

Die Funktion 'pseudoview_update' unterstützt zwei Abgleich-Arten:

1. Beim normalen Datenabgleich wird zunächst eine zweite temporäre Tabelle entsprechend der View-Definition erzeugt. Durch Vergleich der beiden temporären Tabellen kann dann festgestellt werden, welche Datenmanipulationen auf der ersten durchgeführt wurden. Diese Veränderungen werden dann nachgeführt auf die Relation, die für das Durchschreiben der Datenmanipulationen vorgesehen ist.

2. Darüber hinaus hat man bei den Pseudoviews die Möglichkeit, bei der Definition des Views die Option WITH CHECK zu verwenden. Dadurch wird sichergestellt, dass alle Datenmanipulationen den Vorgaben der View-Definition genügen. Dazu wird in der ersten Phase zunächst eine Kopie der Relation erzeugt, auf die die Datenmanipulationen durchgeschrieben werden sollen. Der Abgleich findet dann wie beim normalen Abgleich auf dieser Kopie statt. In der zweiten Phase wird auf Basis der Kopie nocheinmal eine temporäre Tabelle erzeugt, die jetzt im Gegensatz zur ursprünglichen Pseudoview-Tabelle nur noch Tupel enthält, die der View-Definition entsprechen. Mit dieser neuen temporären Tabelle wird ein zweiter normaler Datenabgleich vorgenommen, diesmal jedoch auf der originalen Durchschreibe-Tabelle.

Wie bei den echten Views können auch bei den Pseudoviews die Primärschlüssel-Attribute der Durchschreibetabelle nicht geändert werden.

A Anhang

A.1 Entity-Relationship-Diagramme

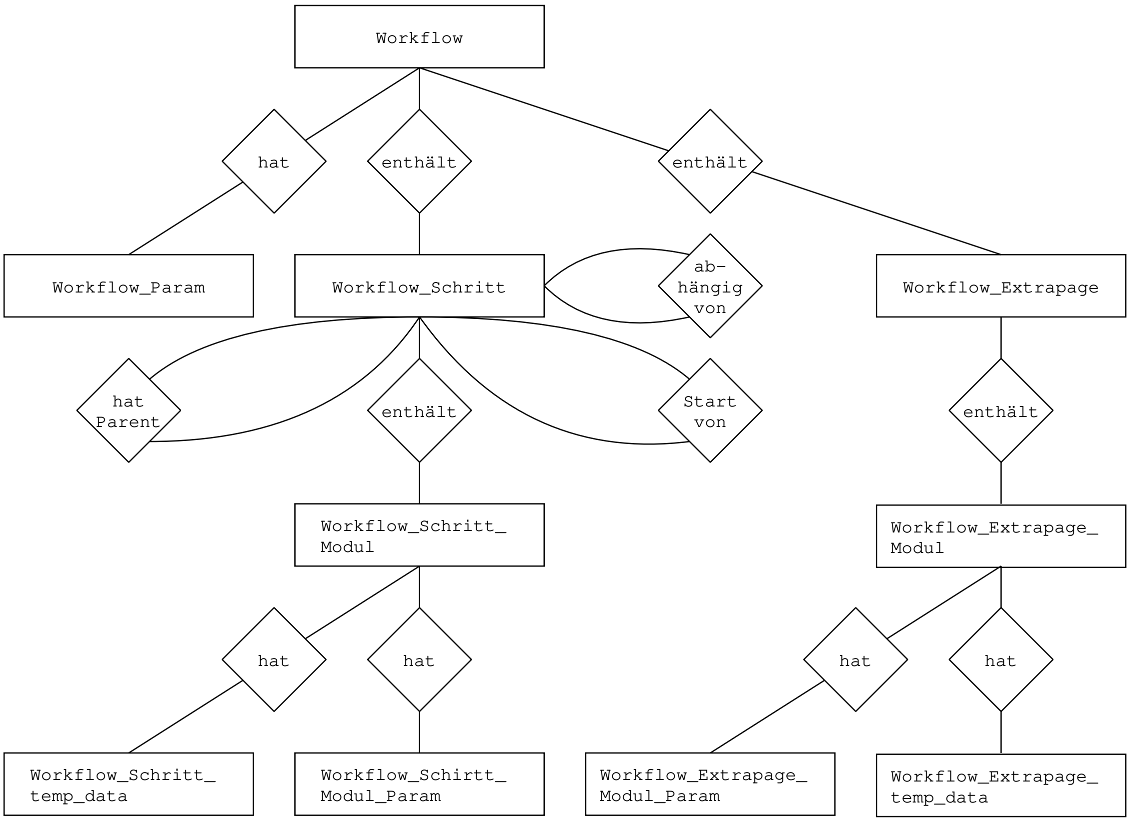

Abbildung 10: ER-Diagramm für Workflow-Steuerung

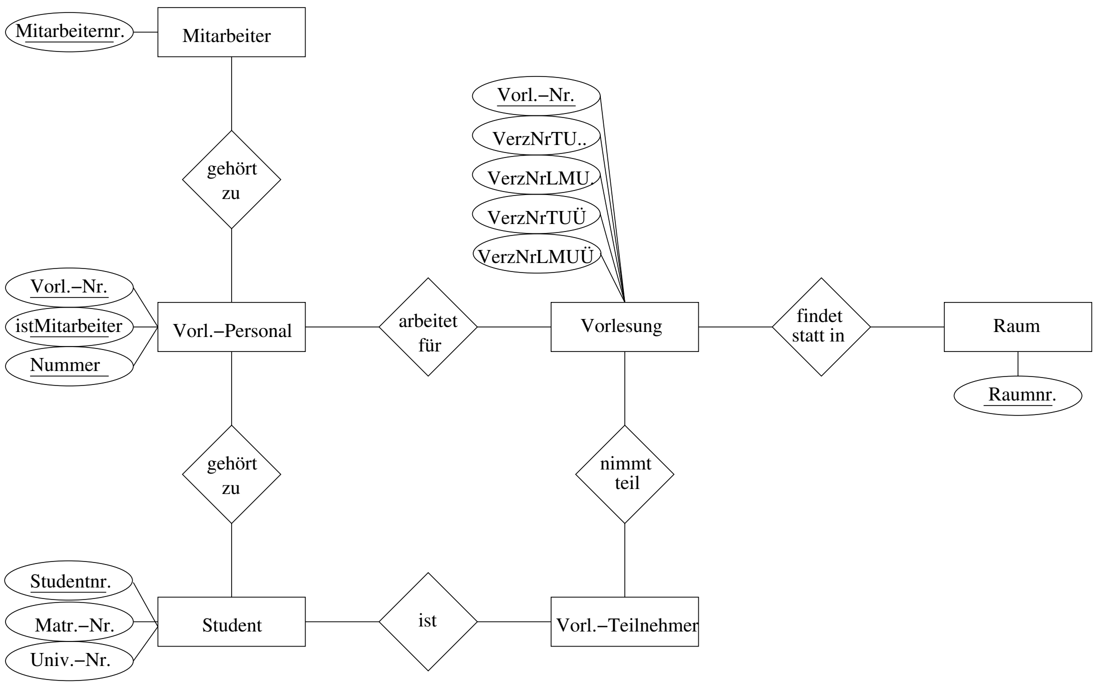

Abbildung 11: ER-Diagramm für Vorlesung

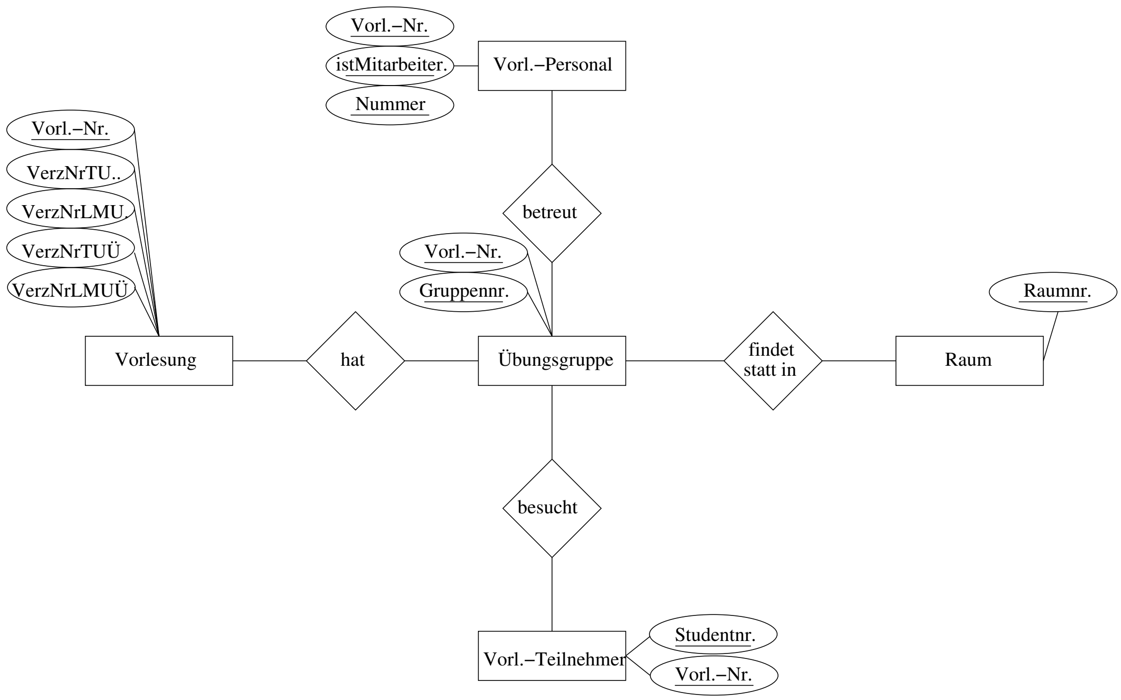

Abbildung 12: ER-Diagramm für Übungsgruppe

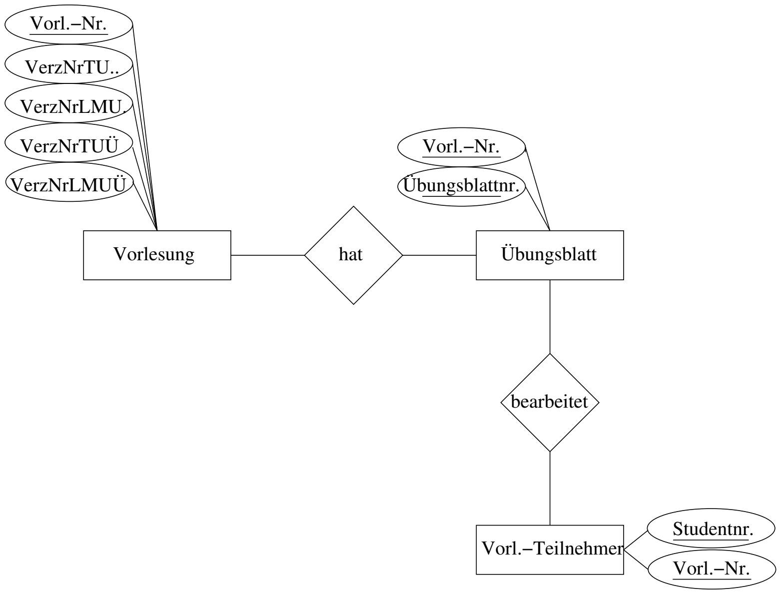

Abbildung 13: ER-Diagramm für Übungsblatt

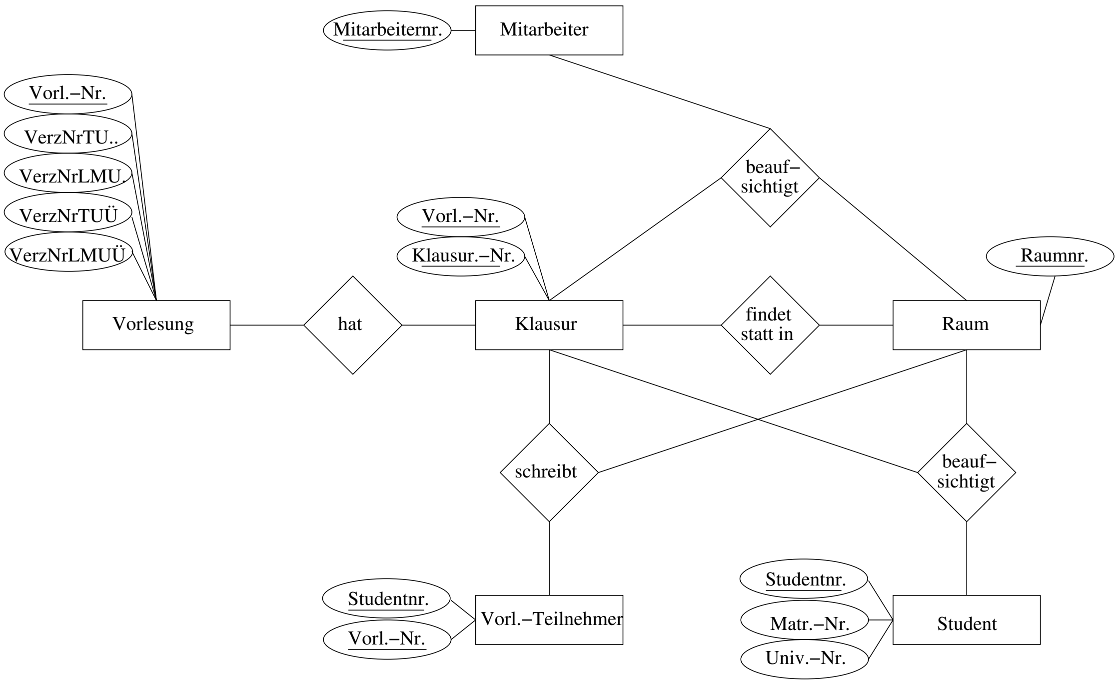

Abbildung 14: ER-Diagramm für Klausur

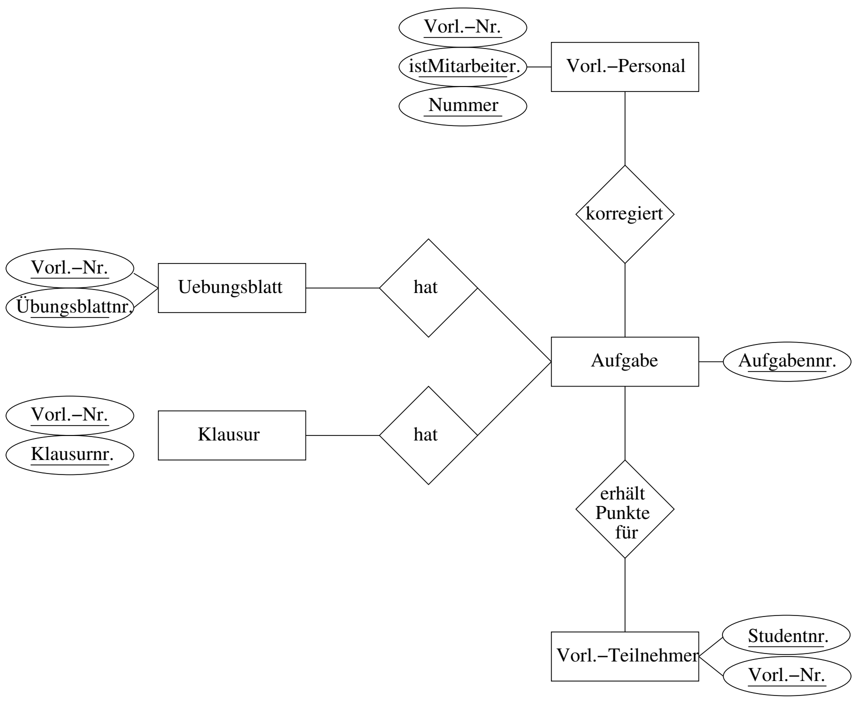

Abbildung 15: ER-Diagramm für Aufgabe

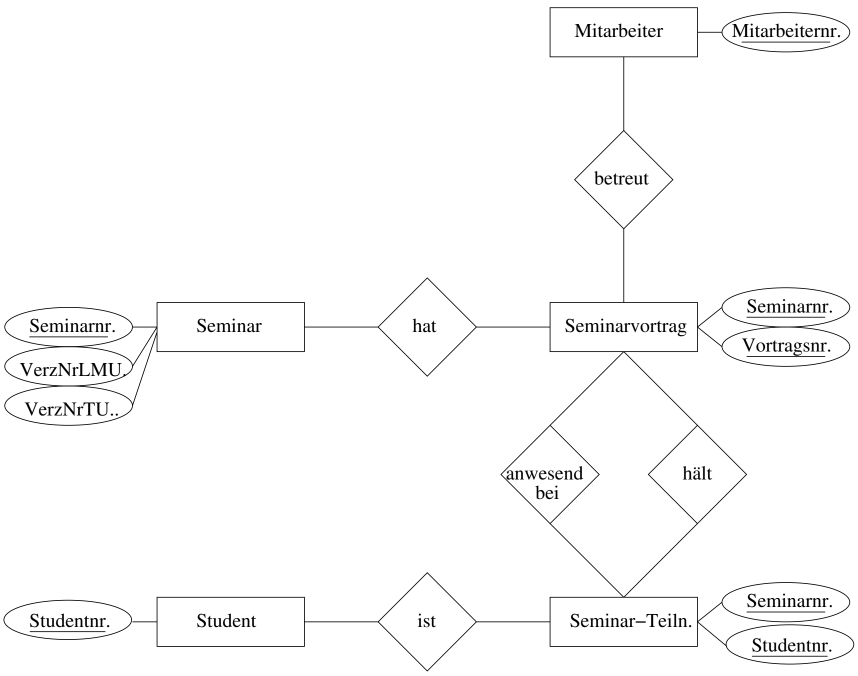

Abbildung 16: ER-Diagramm für Seminar

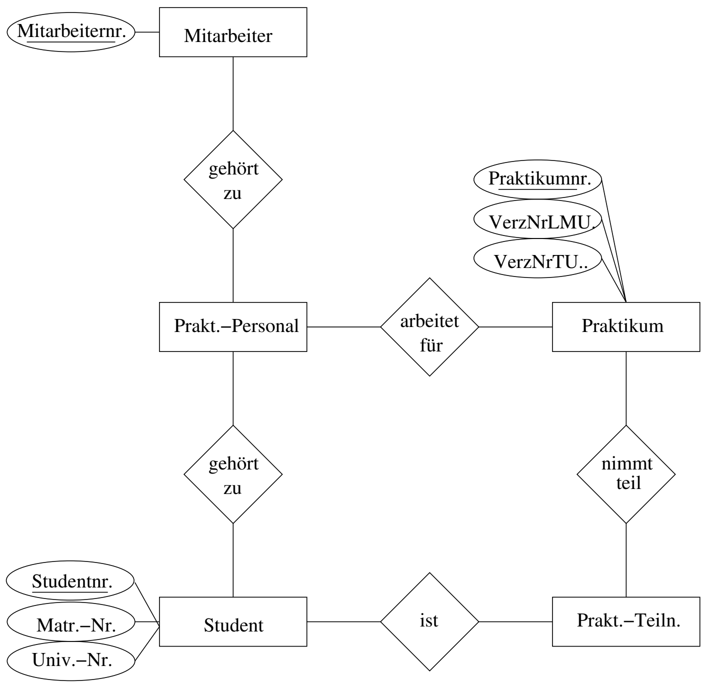

Abbildung 17: ER-Diagramm für Praktikum

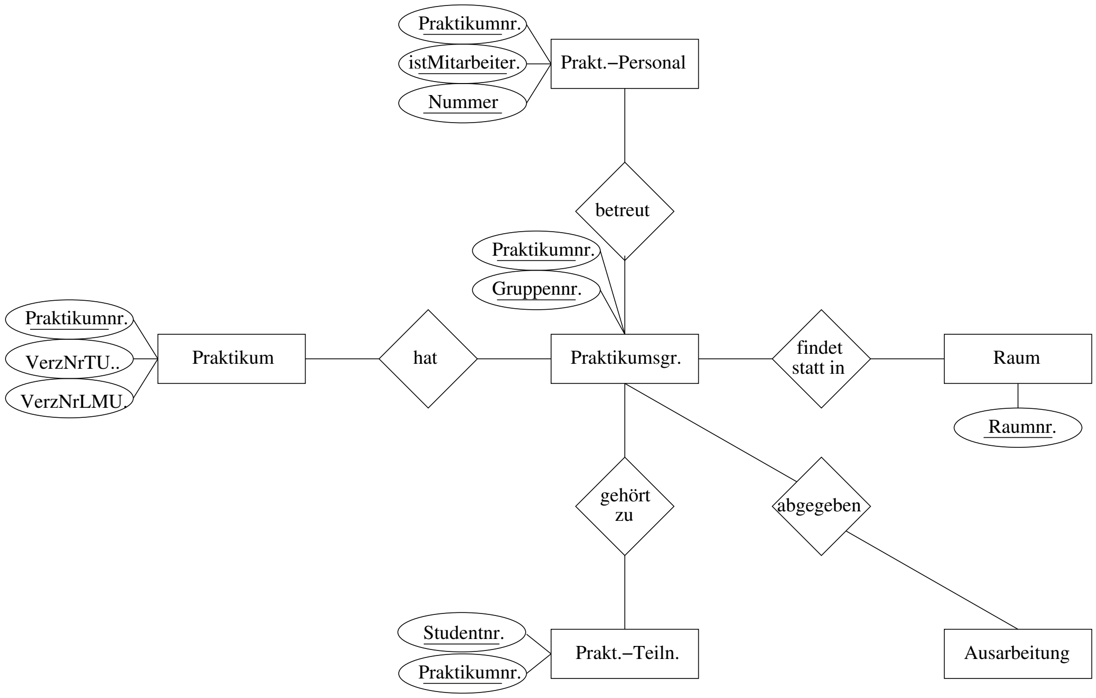

Abbildung 18: ER-Diagramm für Praktikumsgruppe

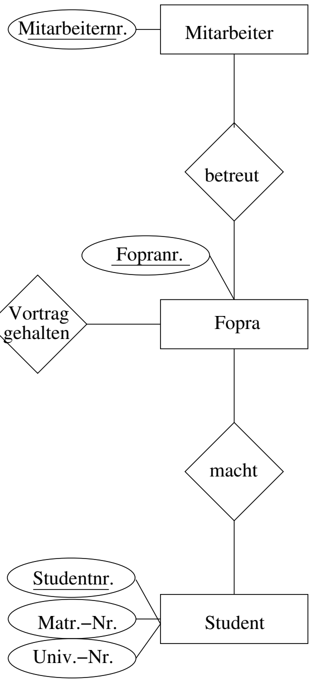

Abbildung 19: ER-Diagramm für Fopra (Diplomarbeit)

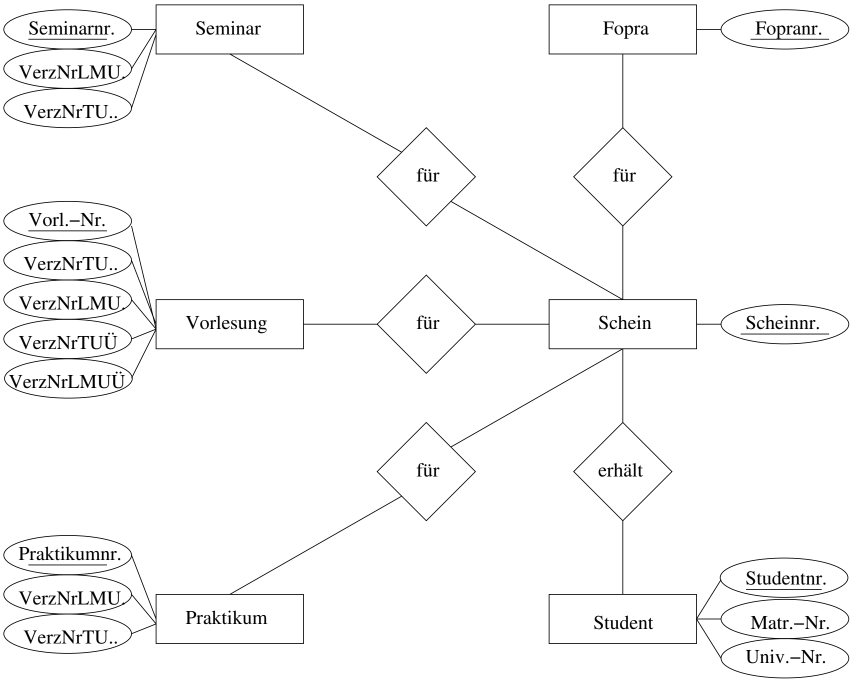

Abbildung 20: ER-Diagramm für Schein