Synchronisation und Integration der zwei Lernplattformen Moodle und MUMIE

Bachelorarbeit

im Fachbereich Wirtschaftswissenschaften II

Internationaler Studiengang Medieninformatik

der

Fachhochschule für Technik und Wirtschaft Berlin

Peter Ritter

Matrikel-Nr: s0512662

Erstprüferin:	Prof. Dr. Debora Weber-Wulff
Zweitprüfer:	Dr. Tilman Rassy

Bearbeitungszeitraum: 08. Oktober 2007 bis 14. Januar 2008

Berlin, Januar 2008

Inhaltsverzeichnis:

1. Einführung

In der Informationsgesellschaft des 21. Jahrhunderts wächst das Wissen der Menschheit in immer kürzeren Intervallen. Dies erfordert neue und schnellere Wege, sich dieses Wissen anzueignen. Der Bereich des e-Learning, der seit Mitte der 90er Jahre vor allem durch die Verbreitung des Internets einen starken Aufschwung erlebte, bietet hierbei nicht nur für (Hoch-)Schulen gute Möglichkeiten, die Präsenzlehre zu unterstützen. Auch in der Erwachsenenbildung und anderen didaktischen Szenarien können die verschiedenen Formen des e-Learning genutzt werden, um Wissen zu vermitteln.

Hierbei haben sich viele verschiedene Formen entwickelt, wie e-Learning genutzt werden kann. Diese reichen von virtuellen Klassenräumen, die genutzt werden, um örtlich getrennte Lehrende und Lernende miteinander zu verbinden und so eine synchrone Form des Lernens zu ermöglichen, über Learning Communities, in denen verschiedene Teilnehmer, die alle gleiche Lerninteressen haben, ihr jeweils individuelles Wissen beisteuern, um so das gemeinsame Wissen zu erweitern, bis hin zu sogenannten e-Collaborations. Dies sind Szenarien, in denen durch webbasierte Zusammenarbeit am Computer die Möglichkeit geschaffen wird, gemeinsam im Team, in Lerngruppen oder mit Kollegen an Präsentationen oder anderen Inhalten kooperativ und zeitgleich zu arbeiten.

Diese verschiedenen Nutzungsformen basieren auf entsprechend unterschiedlichen Technologien. Dies können Learning Management Systeme sein, die hauptsächlich administrative Aufgaben übernehmen und etwa Nutzer, Kurse und Inhalte verwalten, Autorensysteme, die der Erstellung von Inhalten für Lernangebote dienen, oder auch Teleteachingsysteme, welche vor allem durch die Übertragung von Bild und Ton, sowie die Möglichkeit, gleichzeitig an Inhalten arbeiten zu können, eine Möglichkeit darstellen, um e-Collaboration zu betreiben.

Mit den wachsenden technischen Möglichkeiten steigen auch die Ideen und Anforderungen an neue e-Learning Tools. Wenn etwa ein neues System etabliert werden soll, so muss sichergestellt sein, dass dieses sich problemlos in die bereits bestehende e-Learning Landschaft integrieren lässt.

1.1. Motivation

Mathematik ist eine der Schlüsseltechnologien der Gegenwart. Sie bietet nicht mehr bloß die Basis für ingenieur- und naturwissenschaftliche Gebiete, sondern ist in nahezu jedem Bereich unseres gesellschaftlichen Lebens vertreten. Der Umgang mit Zahlen und Statistiken ist ein wichtiges Werkzeug in vielen Situationen. Und obwohl die Mathematik eine der ältesten Wissenschaften darstellt, hat sich unser Wissen über sie doch in den letzten Jahrzehnten rasant vermehrt. Vor dieser Entwicklung stellt sich mehr und mehr die Frage, wie man Lernende möglichst effektiv in diesem Wissen und dem Umgang damit schult.

Das Projekt Multimediale Mathematikausbildung für Ingenieure – kurz MUMIE – wurde im Jahr 2000 an der Technischen Universität Berlin in Zusammenarbeit mit der Technischen Universität München, der Rheinisch-Westfälischen Technischen Hochschule Aachen und der Universität Potsdam konzipiert und von 2001 bis 2004 vom Bundesministerium für Bildung und Forschung (BMBF) gefördert. Resultat dieses Projekts war ein Konzept, das „moderne Technologien sinnvoll in die Präsenzlehre integriert", wie es auf der Projekthomepage[1] heisst. Der Anspruch diese Konzepts war es, das Verständnis von mathematischen Zusammenhängen in den Vordergrund zu stellen, anstatt Wissen nur bereitzustellen. Auch nach Beendigung der Förderung durch das BMBF wurde vor allem in Berlin weiter intensiv an MUMIE gearbeitet und seit 2005 wird diese dort in der Ausbildung für Ingenieure mit größeren Studentenzahlen genutzt.

MUMIE beinhaltet zwar einige Verwaltungsfunktionalitäten, doch stellen diese nicht ihr eigentliches Anliegen dar und sind daher nur soweit implementiert, wie es für das Funktionieren der Plattform nötig ist. MUMIE soll sich auf ihre Kernaufgabe beschränken, das multimediale Lehren und Lernen von Mathematik. Mehr organisatorische Aufgaben sollen ausgelagert werden in ein Learning Management System (LMS), welches in Abschnitt 2.1 genauer beschrieben wird.

Zwar wird MUMIE an der TU Berlin bereits mit dem dort entwickelten LMS Moses verwendet; doch ist es gewünscht, dass MUMIE noch größere Verbreitung findet. Um dies zu erreichen, soll auch ein bereits etabliertes System als Master-LMS fungieren. Die vorliegende Bachelorarbeit wird zeigen, wie eine solche Anbindung als Integration in das LMS Moodle, das in Abschnitt 2.3. beschrieben wird, erfolgt und welche Vorteile dies für die Nutzer bringt.

1 [MUM oJ]

1.2. Anforderungen und Zielsetzung

Das Ziel der vorliegenden Arbeit ist es, MUMIE derart in Moodle zu integrieren, dass das LMS alle administrativen Verwaltungsaufgaben übernimmt. Hierbei wird darauf geachtet werden, dass diese Integration didaktisch sinnvoll in Moodle integriert wird. Ein Nutzer soll keinen Bruch mehr zwischen den beiden Systemen wahrnehmen können. Hierzu sollte das LMS auch eine Portalfunktion für MUMIE bereitstellen, so dass man direkt von Moodle nach MUMIE navigieren kann, ohne dass ein Nutzer sich dort aktiv neu authentifizieren muss. Die gewohnten Arbeitsschritte und Darstellungen in Moodle sollen nicht verändert werden, so dass Nutzer, die bereits Erfahrungen mit diesem LMS haben, die Erweiterungen intuitiv nutzen können. Es ist dabei nicht gewünscht, die Strukturierung der Lerninhalte in MUMIE oder die Navigation zwischen deren Aufgaben innerhalb von Moodle nachzubauen.

Die Implementierung dieser Synchronisation und Integration soll einzig auf Seiten von Moodle stattfinden. Änderungen oder Erweiterungen am Code der MUMIE sind nicht gewünscht. Bei dem Einbau der gewünschten Funktionalitäten ist darauf zu achten, dass diese derart modular in das LMS integriert werden, dass der Code gut wartbar und resistent gegen Änderungen durch Updates bleibt. Dies kann nur erreicht werden, wenn keine Stellen am Kern Code von Moodle verändert oder erweitert werden müssen. Sollte sich im Laufe der Arbeit herausstellen, dass dies nicht ohne weiteres möglich ist, so muss überprüft werden, ob es eine Möglichkeit gibt, die benötigten Erweiterungen des Quellcodes fest in zukünftige Moodle Veröffentlichungen aufnehmen zu lassen. Hierzu ist es unabdingbar, entsprechende Vorhaben ausführlich mit den Hauptentwicklern und der Moodle Gemeinschaft zu kommunizieren.

Die Zusammenarbeit zwischen den beiden Lernplattformen soll letztlich so realisiert werden, dass eine bidirektionale Kommunikation zwischen den Systemen stattfindet. Dabei soll darauf geachtet werden, dass lediglich das LMS Aktionen anstößt. Dies kann geschehen durch das Senden von Informationen oder Informationsanfragen. MUMIE soll auf solche Requests entsprechend reagieren. Alle Daten, die MUMIE vom Master-LMS bezieht, sollen intern rudimentär gespiegelt werden, was bedeutet, dass nur die von MUMIE benötigten Informationen zu einem Objekt in MUMIE gespeichert werden, die dort benötigt werden. In einigen Fällen reicht es, wenn MUMIE lediglich einen Platzhalter bereitstellt und die vollständigen Informationen im LMS ausgelagert bleiben. Gleiches gilt für Informationen wie beispielsweise erreichte Punktzahlen, welche von Moodle aus MUMIE ermittelt werden. Auch diese sollen nicht vollständig im LMS abgebildet werden.

Ein Synchronisationsmechanismus sorgt dafür, dass die Daten zwischen den beiden Plattformen konsistent gehalten werden. Wenn sich Informationen in Moodle ändern oder neue hinzugefügt werden, so soll MUMIE vom LMS automatisch darüber unterrichtet werden. Da MUMIE nicht selbst aktiv werden soll, muss eine Möglichkeit gefunden werden, wie Moodle an Stellen, an denen es relevant sein könnte, Anfragen nach etwaigen Änderungen nach MUMIE schickt, um die entsprechenden Informationen zu ermitteln. Die relevanten Daten, die zwischen den beiden Lernplattformen für diese Integration synchronisiert werden müssen, werden hierbei im folgenden Abschnitt genau aufgelistet werden.

1.2.1. Zu synchronisierende Daten

Die Synchronisation der administrativen Objekte vom LMS nach MUMIE betrifft Semester, Lehrveranstaltungen, Tutorien und Benutzer. Eine Lehrveranstaltung in MUMIE bezieht sich immer auf ein bestimmtes übergeordnetes Semester und beinhaltet mindestens ein eingebettetes Tutorium, über welches die Nutzer zur Lehrveranstaltung zugeordnet werden. Diese Abhängigkeiten sind innerhalb von MUMIE zwingend und müssen somit auch im LMS eingehalten werden. Unter Nutzern werden in MUMIE sowohl Studenten als auch Tutoren, Dozenten und Autoren von Lerninhalten verstanden. Die Synchronisation muss zumindest Studenten, Tutoren und Dozenten erfassen. Von der Synchronisation brauchen allerdings nur diejenigen Benutzer, Tutorien, Lehrveranstaltungen und Semester erfasst zu werden, die MUMIE tatsächlich kennen muss. Entsprechende Objekte anderer Veranstaltungen, die in keinem Bezug zu MUMIE stehen, sollen hier ausgeklammert werden.

Folgende Daten sollen von Moodle aus kontrolliert werden (in Klammern stehen jeweils die Namen der Tabellen oder Spalten der MUMIE Datenbank):

- bei einem Benutzer (user):
  - Nutzername (login_name)
  - verschlüsseltes Passwort (password)
  - Vorname (first_name)
  - Nachname (surname)
  - wenn vorhanden die Matrikelnummer (matr_number)

- bei einem Semester (semester):
  - Name (name)
  - optional eine Kurzbeschreibung (description)
- bei einer Lehrveranstaltung (class):
  - Name (name)
  - optional eine Kurzbeschreibung (description)
  - das übergeordnete Semester (semester)
  - eine Liste von Dozenten (class_lecturers)
  - der jeweils verknüpfte MUMIE-Kurs als Lerninhalt (course)
- bei einem Tutorium (tutorial):
  - Name (name)
  - optional eine Kurzbeschreibung (description)

Für die Fälle, in denen Moodle keine Kurzbeschreibungen zur Verfügung stellt, können diese von der Synchronisation ausgeschlossen werden. In diesen Fällen wird NULL in die entsprechende Spalte in der MUMIE Datenbank eingetragen.

Um beim automatischen Anlegen von administrativen Objekten in MUMIE alle notwendigen Informationen zu haben und für eine nutzerfreundliche Darstellung innerhalb von Moodle ist es nötig, dass auch einige Daten aus MUMIE heraus ermittelt werden. Die zu ermittelnden Daten sind zum einen eine Liste aller in MUMIE vorhandenen Lerninhalte, die mit einer Lehrveranstaltung verknüpft werden können und zum anderen Informationen über mit einem Lerninhalt verbundene Punktzahlen. Hierbei sind sowohl maximal mögliche, wie auch von Studenten zu einem gewissen Zeitpunkt erreichte Punkte zu ermitteln.

1.3.Besonderheiten der Terminologie und Darstellung

Bei einer genaueren Betrachtung der e-Learning Landschaft fällt auf, dass es keine einheitliche Terminologie bezüglich Veranstaltungen und Inhalten gibt. So repräsentiert ein Kurs in MUMIE nicht die Lehrveranstaltung an sich, wie dies etwa in Moodle der Fall ist, sondern den Inhalt dieser. Durch diese inhomogene Nutzung von Begriffen kann schnell Verwirrung aufkommen. Hinzu kommt für die vorliegende Arbeit die Schwierigkeit, dass bestimmte Objekte, die von Moodle aus synchronisiert werden sollen, dort gar keine Entsprechungen haben. Um diese Problematik zu entschärfen, werden im Folgenden die verschiedenen Termini und Darstellungen zwischen den beiden Plattformen gegenübergestellt.

Die für die Synchronisation relevanten (Pseudo-)Objekte auf Seiten von MUMIE sind Semester, Klasse, Tutorium und Nutzer. Hierbei bildet ein Semester den Rahmen für mehrere Klassen, welche die jeweiligen Lehrveranstaltungen darstellen. Innerhalb dieser muss es mindestens ein Tutorium geben. Nutzer werden der Lehrveranstaltung zugeordnet, indem sie in genau einem solchen Tutorium teilnehmen müssen. Auch die Bearbeitung der Kurse, welche die Lerninhalte von Lehrveranstaltungen in MUMIE darstellen, wird jeweils pro Tutorium organisiert.

In Moodle ist es nicht unbedingt erforderlich, einen speziellen Rahmen zu erstellen, in welchem sich die Lehrveranstaltungen befinden. Das System selbst stellt diesen Rahmen dar. Es können aber neue Kurskategorien erstellt werden, in welchen jeweils Mengen von Kursen zusammengefasst werden können. Diese Kurskategorien in Moodle wurden folglich für die Synchronisation als Entsprechung der Semester in MUMIE gewählt. Zwingende Tutorien sind in Moodle nicht nötig, da Nutzer oder auch Aufgaben direkt einer Lehrveranstaltung zugeordnet werden. Innerhalb einer solchen gibt es aber nochmal die Möglichkeit, verschiedene Gruppen zu erstellen, denen die Teilnehmer zugeordnet werden können. Da die Bearbeitung der verschiedenen Lerninhalte in Moodle wie etwa Abgaben auch nach diesen Gruppen separiert werden können, stellen sie das Gegenstück zu den Tutorien in MUMIE dar.

Die verschiedenen Begriffe und Darstellungen zwischen den Objekten der beiden Lernplattformen sind in der folgenden Tabelle noch einmal gegenübergestellt.

MUMIE	***Moodle***
Semester	Kurs-Kategorie
Klasse (Lehrveranstaltung)	Kurs
Tutorium	Gruppe
Nutzer	Nutzer
Kurs	Lerninhalt (hier: MUMIE Aktivität)

Tabelle 1: Tabellarische Gegenüberstellung der verwendeten Begriffe in MUMIE und Moodle

2. Analyse der vorhandenen Systeme

Um die Integration der beiden Lernplattformen den Anforderungen entsprechend realisieren zu können, ist es nötig, die beiden Systeme mitsamt ihrer Techniken und Erweiterungsmöglichkeiten ausreichend zu kennen. Dies soll in diesem Abschnitt erreicht werden. Hierbei wird zuerst erklärt werden, wie ein Lernmanagementsystem definiert ist und welche Funktionsbereiche es ausmachen.

Im Anschluss werden die beiden zu synchronisierenden Systeme genauer betrachtet werden. Hierbei wird zuerst MUMIE im Kern des Interesses stehen. Im Anschluss daran wird dann das LMS Moodle genauer beschrieben werden, wobei ein Schwerpunkt auf den Erweiterungsmöglichkeiten liegen wird, die als Lösungsalternativen für die Synchronisation und Integration zur Verfügung stehen.

2.1. Das Lernmanagementsystem (LMS)

Ein Lernmanagementsystem oder Learning Management System– kurz LMS – ist ein Softwaresystem, welches meist den Kern einer komplexen webbasierten e-Learning Infrastruktur bildet[2]. Es dient der Verwaltung von Lehraktivitäten mit dazugehörigen Nutzern und Materialien. Eine Lehraktivität kann in diesem Fall eine Lehrveranstaltung aus der Präsenzlehre ergänzen oder etwa einen Rahmen bieten, in welchem sich Teilnehmer Wissen selbst aneignen. Diese Arbeit kann einzeln oder auch kollaborativ mit anderen Teilnehmern erfolgen. Auch die Kommunikation zwischen den Lehrenden und Lernenden spielt eine große Rolle. Primär richten sich solche Lernmanagementsysteme an Schüler und Studenten; sie werden aber auch vermehrt von Unternehmen in der Erwachsenenbildung eingesetzt.

In der Regel zeichnet sich ein Lernmanagementsystem dadurch aus, dass es unter einer homogenen zentralen Oberfläche die folgenden Funktionalitäten vereint[3]:

- Eine Benutzerverwaltung einschließlich der Verwaltung verschiedener Rollen wie etwa Kursersteller, Dozenten oder Studenten

2 Vgl. nach [ete 06]
3 Nach [Sch 03]

- Eine Lehrveranstaltungsverwaltung mit der Verwaltung der dazugehörigen Teilnehmer und Materialien
- Die Darstellung der Lehrveranstaltungen mitsamt der Lerninhalte und Materialien im Browser, meist inklusive einer Möglichkeit zum Download
- Die Bereitstellung verschiedener Lernwerkzeuge wie etwa Aufgaben oder Wikis
- Die Bereitstellung von Autorentools zum Erstellen von Lerninhalten
- Verschiedene Kommunikationsmöglichkeiten wie Foren oder Chats
- Die Verwaltung der von Studenten erbrachten Leistungen einschließlich diverser Auswertungsmöglichkeiten

2.2. MUMIE

MUMIE ist eine Lernplattform, die speziell auf die Vermittlung mathematischer Inhalte ausgelegt ist. Ihr liegt der Anspruch zugrunde, einen Raum zu schaffen, in dem das Verständnis von Zusammenhängen im Vordergrund steht, anstatt mathematisches Wissen lediglich durch eine Sammlung an Sätzen und Definitionen zu lehren. Hierzu stellt sie eine Reihe spezifischer Tools und Features zur Verfügung. Dies sind etwa[4]:

- eine modularisierte vernetzte Wissensbasis
- Visualisierungen
- interaktive Applets
- nichtlineare Navigationskonzepte

Der Java Application Server (JAPS) stellt den serverseitigen Teil von MUMIE dar. Daneben hat die Plattform noch andere Elemente wie etwa die Autorenwerkzeuge oder die Mathletfactory, die zur vereinfachten Entwicklung von Applets dient. Im Blickpunkt dieser Arbeit allerdings steht JAPS .

Technisch basiert JAPS auf Java-Servlet- und XML-Technologie. Die Architektur besteht aus einem Apache Webserver, einem Tomcat Servlet-Container, einem durch MUMIE-eigene Software erweiterten Cocoon Servlet und einer PostgreSQL Datenbank.

4 Vgl. nach [MUM oJ]

Neben den oben genannten Hilfsmitteln zur Wissensvermittlung sind andere, mehr organisatorische Funktionalitäten lediglich rudimentär oder gar nicht implementiert, so dass sich MUMIE komplementär zu den üblichen generischen LMS verhält. Dahinter steckt eine bewusste Designentscheidung. MUMIE soll sich auf die Wissensvermittlung als Kernfunktionalität beschränken und die organisatorischen Funktionalitäten auslagern in ein Master-LMS. Die Lernplattform wurde von vorneherein für die Zusammenarbeit mit einem LMS konzipiert und wird auch so betrieben. An der Technischen Universität Berlin etwa übernimmt den Part das rudimentäre LMS Moses, welches dort in Eigenarbeit entwickelt wurde.

Um aber eine weitere Verbreitung von MUMIE erreichen zu können, ist es gewollt, auch andere LMS diese Aufgabe übernehmen zu lassen. Die technischen Möglichkeiten zur Kommunikation zwischen MUMIE und dem LMS basieren auf entsprechenden Befehlen, die über HTTP übertragen werden. Neben dieser gibt es auch noch weitere Möglichkeiten, die in Abschnitt 3 beschrieben und gegeneinander abgewägt werden.

2.3. Moodle

Moodle ist ein plattformunabhängiges Lernmanagementsystem, das als freie Software unter GNU Public License zur Verfügung gestellt wird. Voraussetzung zum Betrieb von Moodle ist ein Webserver, der PHP verarbeiten kann und eine Datenbank wie etwa MySQL oder PostgreSQL. Ursprünglich war der Name Moodle ein Akronym für Modulare dynamische objektorientierte Schulungsumgebung (Modular Object-Oriented Dynamic Learning Environment). Der Name steht aber heute ausserdem für „eine einfache Lerntechnik [und] das Kennenlernen von neuen Lernmethoden“, so die Moodle Dokumentationsseiten[5].

Moodle stellt sogenannte Kursräume zur Verfügung, in denen Teilnehmer auf Materialien und Lernaktivitäten zurückgreifen oder durch Foren und dergleichen miteinander oder mit den Dozenten kommunizieren können. Arbeitsmaterialien sind Texte, Links oder Dateien, die einer passiven Informationsaufnahme dienen, während Lernaktivitäten eine Interaktion mit dem Lernenden verlangen. Diese Lernaktivitäten sind so modular in das LMS eingegliedert, dass zusätzliche Aktivitäten einfach in eine bestehende Moodle Instanz eingefügt werden können. Diese Aktivitätenmodule bieten

5 [MoD 07]

somit eine der Möglichkeiten zur Erweiterung von Moodle, die im kommenden Abschnitt beschrieben werden.

Moodle bietet durch seinen offenen Quellcode und die recht hohe Verbreitung von etwa 38.000 registrierten Installationen weltweit (Stand Januar 2008)[6] eine gute Basis zum Einbau und der anschließenden Akzeptanz und Verbreitung der Synchronisation mit MUMIE. Aus diesem Grund wurde es ausgewählt, als Master-LMS für MUMIE zu fungieren.

2.3.1. Erweiterungsmöglichkeiten

Um der Anforderung gerecht zu werden, die Synchronisation mit MUMIE so in Moodle zu integrieren, dass der Kern Code des LMS nicht verändert wird, ist es erforderlich, ausreichende Kenntnis über die verschiedenen Erweiterungsmöglichkeiten zu gewinnen. In diesem Abschnitt werden nun die drei Alternativen beschrieben, die dazu in Frage kommen.

2.3.1.1. Block

Blöcke befinden sich örtlich gesehen an den Seiten eines virtuellen Kursraumes und bieten weitreichende Möglichkeiten, Informationen einzubinden. Als Beispiele hierfür sind etwa ein Kalender, letzte Änderungen oder die momentan angemeldeten Nutzer zu nennen. Zusätzlich zu diesen standardmäßig in Moodle enthaltenden Blöcken ist es möglich, mit relativ geringem Aufwand weitere Blöcke hinzuzufügen.

Ein Block dient im Rahmen des Kursraumes dazu, den Teilnehmern Informationen darzustellen. Ein interaktiver Aspekt liegt ihm allerdings nicht zugrunde. Da aber die Integration von MUMIE in Moodle auch darauf zielt, dass sich Studenten auf die eigentlichen Seiten der MUMIE begeben, um etwa die dortigen Aufgaben zu bearbeiten, scheint die Möglichkeit, diese Integration über einen Block zu realisieren, didaktisch nicht sinnvoll zu sein.

6 Nach [Moo oJ]

2.3.1.2. Kursformat

Ein Kursformat beschreibt die Art und Weise, wie ein Kurs strukturiert ist. Die Standardformate von Moodle hierbei sind:

- wöchentlich – Das bedeutet, dass von Woche zu Woche neue Themen, Inhalte oder Aufgaben angeboten werden.
- thematisch – Dies ähnelt sehr dem Wochenformat, nur dass hierbei jeder neue Bereich einem neuen Thema anstatt einer neuen Woche zuzuordnen ist.
- sozial – Dieses Format orientiert sich an einem zentralen Forum und muss nicht unbedingt eine Lehrveranstaltung repräsentieren.

Zu diesen Standardformaten kamen mit der Zeit einige neue Kursformate wie etwa das LAMS-Format. LAMS (Learning Activity Management System)[7] ist ein externes Tool, um online gemeinschaftliche Lernaktivitäten anzubieten und zu managen. Somit besteht eine gewisse Ähnlichkeit zwischen LAMS und MUMIE. Das Kursformat LAMS übernimmt nun solche Lernprozesse von der externen Plattform und strukturiert nach diesen den Kurs. Im Vergleich zu Blöcken erscheint also die Möglichkeit, die Integration von MUMIE als Kursformat zu realisieren, weitaus besser geeignet, da hier die Aktivitäten der Studierenden im Vordergrund stehen. Allerdings hätte dies zur Folge, dass der Aufbau des Kursraumes stark dem Aufbau des Lerninhalts in MUMIE ähneln würde. Eine solche Nachbildung der Navigation zwischen Aufgaben innerhalb von Moodle aber ist laut Anforderungen nicht gewünscht. Statt dessen soll das Wochenformat, welches den Standard in Moodle darstellt, auch für Lehrveranstaltungen, die mit MUMIE synchronisiert werden, genutzt werden.

7 [LAM 02]

2.3.1.3. Lernaktivität

Lernaktivitäten können innerhalb eines Kursraumes nach Belieben zusammengestellt werden und haben die Interaktion mit den Teilnehmern der Lehrveranstaltung als Ziel. Das bedeutet, dass ein Lernender selbst aktiv werden muss. Beispiele hierfür sind aus der Menge der standardmäßig in Moodle enthaltenen Lernaktivitäten etwa das Wiki zum kollaborativen Schreiben von Texten, Foren für Konversationszwecke oder Aufgaben, welche die Teilnehmer innerhalb eines bestimmten Zeitraums bearbeiten müssen. Da sich diese Aktivitäten in Kursräume jeden Kursformats einfügen lassen und auch neue Lernaktivitäten sehr einfach und modular in bereits bestehende Moodle Instanzen integriert werden können, wurde entschieden, auch die in dieser Arbeit behandelte Integration von MUMIE in Moodle als solche Lernaktivität zu realisieren. Die genaue Umsetzung wird in Kapitel 4 genau beschrieben werden.

3. Die Kommunikationsmöglichkeiten

Um nun die Synchronisation der benötigten (Pseudo-)Objekte zwischen Moodle und MUMIE bewerkstelligen zu können, bedarf es einer geeigneten Kommunikationsform zwischen den beiden Systemen. Die Tatsache, dass Moodle und MUMIE mit unterschiedlichen Techniken realisiert wurden, stellt hierbei eine besondere Schwierigkeit dar. Im Laufe der Untersuchung, welche Möglichkeiten hierfür grundsätzlich zur Verfügung stehen, haben sich drei Optionen heraus kristallisiert, auf welche im Folgenden eingegangen wird.

3.1. Web-Services

Da beide Plattformen nicht notwendigerweise auf der gleichen physischen Maschine laufen, sehr wohl aber über HTTP erreichbar sein müssen, wäre es durchaus denkbar, dass MUMIE zur Synchronisation entsprechende Web-Services anbieten könnte, welche von Moodle aufgerufen würden. Dies hätte auch den großen Vorteil, dass andere LMS, welche zu einem späteren Zeitpunkt oder an einer anderen Institution etwa als Master-LMS eingesetzt werden sollen, ebenfalls diese Web-Services nutzen könnten. Allerdings wäre für diese Lösung ein größerer Aufwand auf beiden Systemen zu betreiben, als der Rahmen dieser Bachelorarbeit zulässt. Dies ist vor allem dadurch bedingt, dass vor der Implementierung solcher Web-Services gründliche Überlegungen zu Fragen der Sicherheit und Vertraulichkeit der zu synchronisierenden Daten nötig sind. Des weiteren wären auf beiden Systemen Implementierungsschritte notwendig, was nicht gewünscht ist. Lediglich auf Seiten von Moodle soll Code entwickelt werden. Dieser soll die vorhanden Möglichkeiten zum Austausch oder Erhalt von Informationen der MUMIE nutzen. Somit lag die Entscheidung nun zwischen den beiden folgenden Möglichkeiten.

3.2. Einchecken von Archiven

Die MUMIE bietet die Möglichkeit, durch das Einchecken von Zip-Archiven Inhalte in die Datenbank schreiben zu können. Hierbei wird für jede neue Zeile, die in einer Tabelle in der Datenbank eingetragen werden soll, eine XML-Datei mit den entsprechenden Informationen erstellt. Die Dateien, welche einer festgelegten Ordnerstruktur entsprechen müssen, werden dann in ein Archiv gepackt. Dieses wird nach erfolgreicher Authentifizierung an der MUMIE schließlich als Anhang per HTTP an diese geschickt. Dort wird das Archiv dekomprimiert und die Informationen innerhalb einer Transaktion eingecheckt.

Neben der Tatsache, dass für dieses Vorgehen keine Änderungen oder Erweiterungen am Code der MUMIE nötig sind, liegt der Vorteil dieser Alternative darin, dass die Informationen, welche in die Datenbank eingetragen werden sollen, immer vollständig in diese übernommen werden, da die Menge an einzutragenden Objekten in einer einzigen Transaktion abgearbeitet wird. Sollte innerhalb dieser Transaktion ein Fehler auftauchen, so wird diese abgebrochen und die bereits getätigten Einträge verworfen.

Der große Nachteil dieser Möglichkeit zur Synchronisation von Informationen liegt allerdings darin, dass beim Erstellen der XML-Inhalte des Archivs auf Seiten des Master-LMS eine festgelegte Struktur eingehalten werden muss. Sollte sich diese bei einer späteren Version der MUMIE ändern, so wären auch Änderungen am Code des LMS nötig, welches die XML-Dateien erzeugt.

3.3. HTTP-Requests

Neben der gerade beschriebenen Möglichkeit, Informationen per Checkin in die Datenbank zu bekommen, können entsprechende Requests auch per Synchronisationsbefehl über HTTP geschickt werden. Hierbei authentifiziert sich das Master-LMS bei MUMIE als Client mit bestimmten Rechten und schickt die zu synchronisierenden Informationen als eingebetteten Text in einem POST-Request. MUMIE verarbeitet diese Informationen abhängig vom genutzten Synchronisationsbefehl, erstellt oder ändert die zu synchronisierenden Daten und sendet eine entsprechende Antwort über Erfolg oder Misserfolg zurück an den Sender.

Auch bei dieser Alternative sind keine Erweiterungen am Code von MUMIE nötig. Des weiteren ist es nicht nötig, auf Seiten des Master-LMS besondere Kenntnisse über die Tabellen der MUMIE Datenbank zu haben. Lediglich die Synchronisationsbefehle mit entsprechenden Parametern müssen bekannt sein. Die weitere Verarbeitung der Informationen wird einzig von MUMIE vorgenommen. Wird dort etwas an der Datenbank oder der Verarbeitung geändert, so hat dies keine Folgen für das Master-LMS.

Auf der Basis des Wissens um diese Vorteile und der Tatsache, dass auch schon die rudimentäre Synchronisation zwischen MUMIE und Moses von diesen Synchronisationsbefehlen Gebrauch gemacht hat, ist auch in diesem Fall die Entscheidung getroffen worden, die Synchronisation mit Moodle ebenfalls über eben diese Befehle zu realisieren, die im folgenden genauer Beschrieben werden sollen. Zusätzlich zu den Befehlen, mit denen Inhalt geschrieben wird, werden für die Synchronisation der beiden Lernplattformen auch GET-Requests an MUMIE geschickt, um Informationen zu erhalten und in Moodle darstellen zu können. Im Anschluss an die Beschreibung der Synchronisationsbefehle werden diese Requests noch erläutert werden.

3.3.1. Die Synchronisationsbefehle

Um eine Synchronisation von Daten von Moodle nach MUMIE vorzunehmen, schickt Moodle einen HTTP-Request mit dem entsprechenden Synchronisationsbefehl und den dazugehörigen Parametern als POST an MUMIE. Die URL hat hierbei die folgende Form:

`url_prefix/protected/sync/command`

Hierbei steht der `url_prefix` für den Prefix des Servers, auf welchem die MUMIE Instanz läuft (z. B. `http://www.mumie.net/cocoon`). ImPfad `/protected/sync` wird schließlich der jeweilige Synchronisationsbefehl angestoßen. Er gibt die auszuführende Operation an. Die Daten werden als Parameter übersandt. Um die zu synchronisierenden Daten zwischen den Systemen eindeutig identifizieren zu können, werden sogenannte Synchronisations-IDs eingeführt und bei jedem Aufruf eines Synchronisationsbefehls mitgeschickt. Eine solche ID ist ein String, der das zu jeweilige (Pseudo-)Objekt eindeutig bestimmt. Dieser String setzt sich zusammen durch die folgende Konvention: Zuerst nennt er den Namen des Master-LMS gefolgt von dem Namen derjenigen Tabelle im LMS, welche das zu synchronisierende Objekt beinhaltet und anschließend der ID des Objekts

innerhalb dieser Tabelle. Die verschiedenen Angaben werden durch einen Bindestrich separiert (z. B. `moodle-mdl_user-3`).

Als Antwort auf einen solchen Synchronisationsbefehl schickt MUMIE ein Text-Dokument. War der Synchronisationsvorgang erfolgreich, so besteht das Dokument aus dem String "OK". Trat ein Fehler auf, besteht das Dokument aus einem String mit der jeweiligen Fehlermeldung.

Die einzelnen Synchronisationsbefehle mit ihren Übergabeparametern werden im Folgenden detailliert aufgelistet:

- **new-user**

  Dieser Befehl legt einen neuen Benutzer an. Die Request-Parameter sind:

  - sync-id: Die Synchronisations-ID des Nutzers
  - login-name: Der Benutzername
  - password-encrypted: Das gehashte Passwort des Nutzers
  - first-name: Der Vorname des Nutzers
  - surname: Der Familienname des Nutzers
  - matr-number: Die Matrikelnummer des Nutzers. Dieser Parameter ist optional.

- **change-user-data**

  Dieser Befehl ändert die Daten eines bestehenden Nutzers. Die Request-Parameter sind:

  - sync-id: Die Synchronisations-ID des Nutzers
  - login-name: Der Benutzername
  - password-encrypted: Das gehashte Passwort des Nutzers
  - first-name: Der Vorname des Nutzers
  - surname: Der Familienname des Nutzers
  - matr-number: Die Matrikelnummer des Nutzers.

  Ausser der sync-id müssen nur die Parameter gesendet werden, die geändert werden sollen.

- **new-class**

  Dieser Befehl legt eine neue Lehrveranstaltung an. Die Request-Parameter sind:

  - sync-id: Die Synchronisations-ID der Lehrveranstaltung
  - name: Der Name der Lehrveranstaltung
  - descripion: Eine Kurzbeschreibung der Lehrveranstaltung als Text. Dieser Parameter ist optional.
  - semester: Die Synchronisations-ID des übergeordneten Semesters.
  - lecturers: Die Synchronisations-IDs der Dozenten der Lehrveranstaltung durch Leerzeichen und/oder Komma getrennt.
  - courses: Die IDs der Lerninhalte (MUMIE-Kurse) aus der MUMIE, welche mit dieser Lehrveranstaltung verknüpft werden sollen.

- **change-class-data**

  Dieser Befehl ändert die Daten einer bestehenden Lehrveranstaltung. Die Request-Parameter:

  - sync-id: Die Synchronisations-ID der Lehrveranstaltung
  - name: Der Name der Lehrveranstaltung
  - descripion: Eine Kurzbeschreibung der Lehrveranstaltung als Text.
  - semester: Die Synchronisations-ID des übergeordneten Semesters.
  - lecturers: Die Synchronisations-IDs der Dozenten der Lehrveranstaltung durch Leerzeichen und/oder Komma getrennt.

  Ausser der sync-id müssen nur die Parameter gesendet werden, die geändert werden sollen.

- **new-tutorial**

  Dieser Befehl legt ein neues Tutorium an. Die Request-Parameter sind:

  - sync-id: Die Synchronisations-ID des Tutoriums
  - name: Der Name des Tutoriums
  - description: Eine Kurzbeschreibung. Dieser Parameter ist optional.
  - tutor: Die Synchronisations-ID des Tutors dieses Tutoriums
  - class: Die Synchronisations-ID der übergeordneten Lehrveranstaltung

- **change-tutorial-data**

  Dieser Befehl ändert die Daten eines bestehenden Tutoriums. Die Request-Parameter sind:

  - sync-id: Die Synchronisations-ID des Tutoriums
  - name: Der Name des Tutoriums
  - description: Eine Kurzbeschreibung.
  - tutor: Die Synchronisations-ID des Tutors dieses Tutoriums
  - class: Die Synchronisations-ID der übergeordneten Lehrveranstaltung

  Ausser der sync-id müssen nur die Parameter gesendet werden, die geändert werden sollen.

- **new-semester**

  Dieser Befehl legt ein neues Semester an. Die Request-Parameter sind:

  - sync-id: Die Synchronisations-ID des Semesters
  - name: Der Name des Semesters
  - descripion: Eine Kurzbeschreibung des Semesters. Dieser Parameter ist optional.

- **change-semester-data**

  Dieser Befehl ändert die Daten eines bestehenden Semesters. Die Request-Parameter sind:

  - sync-id: Die Synchronisations-ID des Semesters
  - name: Der Name des Semesters
  - description: Eine Kurzbeschreibung des Semesters

  Ausser der sync-id müssen nur die Parameter gesendet werden, die geändert werden sollen.

- **add-user-to-tutorial**

  Dieser Befehl fügt einen bestehenden Studenten einem Tutorium hinzu.

  Die Request-Parameter sind:

  - tutorial: Die Synchronisation-ID des Tutoriums
  - user: Die Synchronisations-ID des Studenten

- **remove-user-from-tutorial**

  Dieser Befehl entfernt einen Studenten aus einem Tutorium. Die Request-Parameter sind:

  - tutorial: Die Synchronisations-ID des Tutoriums
  - user: Die Synchronisations-ID des Studenten

3.3.2. Die GET-Requests

Zusätzlich zu diesen Synchronisationsbefehlen, welche dazu genutzt werden, Informationen von Moodle nach MUMIE zu schreiben, bedient sich die vorliegende Integration von Moodle und MUMIE bestimmter GET-Requests, mit denen Informationen aus MUMIE geholt werden, um diese in Moodle darzustellen. Diese Requests werden nun genauer beschrieben.

- **Informationen zu Lerninhalten**

Da in MUMIE die Lerninhalte (also die MUMIE Kurse) immer mit einer jeweiligen Lehrveranstaltung verknüpft werden, ist es nötig, diese beiden (Pseudo-)Objekte an geeigneter Stelle miteinander zu verbinden. Um diese Verbindung auch aus Moodle heraus anstoßen zu können, wird nach der Authentifizierung des LMS als Client bei MUMIE ein GET-Request an die URL

`/cocoon/protected/data/document-index/type-name/course` geschickt. Als Ergebnis antwortet MUMIE mit einem im HTTP-Response eingebetteten XML Dokument, aus dem sich nach entsprechender Analyse alle nötigen Informationen zu den bereits existierenden MUMIE Kursen auslesen lassen.

- **Informationen zu Punktzahlen**

Um auch innerhalb von Moodle Informationen über maximal erreichbare und aktuell vorhandene Punkte zu einer Lehrveranstaltung mit MUMIE Aktivität darstellen zu können, ist es nötig, auch diese Informationen aus MUMIE zu ermitteln. Hierzu wird wieder nach der Authentifizierung des LMS als Client bei MUMIE ein GET-Request an die URL

`/cocoon/protected/data/total-class-grades` geschickt. Diesem Request wird zusätzlich als Parameter die Synchronisations-ID der übergeordneten Lehrveranstaltung angehängt. Als Antwort sendet MUMIE wieder ein im HTTP-Response eingebettetes XML Dokument mit den entsprechenden Informationen, so dass Moodle diese für den Nutzer darstellen kann.

3.3.3. Authentifizierung des Master-LMS bei MUMIE

Wie bereits erwähnt wurde, tritt Moodle gegenüber MUMIE als HTTP Client auf, um Zugang zu den geschützten URLs zu bekommen und so die Synchronisation anzustoßen oder Informationen zu erhalten. Als Voraussetzung hierzu muss das LMS als Nutzer in MUMIE eingetragen sein und über die notwendigen Synchronisationsrechte verfügen.

Um diese Authentifizierung und die folgenden GET- und POST Aufrufe zu automatisieren, bedient man sich einer speziellen Klasse – dem JapsClient – welche die entsprechenden Schritte wie das

Anmelden und den Umgang mit Cookies übernimmt. Alle POST und GET Methoden liefern als Ergebnis eine HttpURLConnection, die im weiteren Verlauf von MUMIE genutzt wird, um den jeweiligen Request zu bearbeiten und mit einem entsprechenden Response zu antworten.

Da sowohl die laufende rudimentäre Synchronisation zwischen Moses und MUMIE als auch die Applets der MUMIE selbst diesen JapsClient nutzen, um Informationen in MUMIE einzutragen, ist dieser bereits Bestandteil des gesamten MUMIE Pakets. Für die Nutzung dieser in Java implementierten Klasse mit Moodle war es allerdings nötig, die Logik in PHP umzuschreiben.

4. Umsetzung

Nachdem nun also die wichtigen Vorentscheidungen zur Art des Einbaus in Moodle und zur Verwendeten Kommunikationsmöglichkeit zwischen den beiden Systemen getroffen sind, geht es um die konkrete Umsetzung der Implementierung. Diese, die dabei auftretenden Schwierigkeiten und Lösungen werden in diesem Abschnitt erläutert werden. Die drei großen Abschnitte, die dabei genauer beschrieben werden, sind das Design der zusätzlichen Tabellen innerhalb der Moodle Datenbank, der Aufbau der MUMIE-Aktivität in Moodle – hierbei soll unterschieden werden zwischen interner Kommunikation und externer Kommunikation mit MUMIE – und aufgetretene Darstellungs- und Repräsentierungsprobleme zwischen den beiden Systemen.

4.1. Design der zusätzlichen Datenbanktabellen in Moodle

Für jedes neue Aktivitätenmodul in Moodle gibt es die Möglichkeit, zusätzliche Tabellen in die existierende Datenbank hinzuzufügen, um bestimmte Informationen zu speichern. Hierfür definiert man an festgelegter Stelle per XML die Struktur dieser Tabellen. Beim Einbinden des neuen Moduls in eine Instanz von Moodle werden die Tabellen dann automatisch angelegt. Für die hier beschriebene MUMIE Aktivität wurden die beiden folgenden Tabellen definiert:

Spaltenname	*Typ*	*Beschreibung*
id	bigint(10)	ID der Tabelle (Primärschlüssel)
course	bigint(10)	ID der Lehrveranstaltung, in welcher die Aktivität existiert.
name	varchar(255)	Name der Aktivität
description	text	Kurzbeschreibung (optional)
grade	bigint(10)	Maximal erreichbare Punktzahl pro Aktivität
timemodified	bigint(10)	Zeitpunkt der letzten Änderung (UNIX-Format)

Tabelle 2: mumiemodule – Informationen zu der entsprechenden MUMIE-Aktivität

Spaltenname	*Typ*	*Beschreibung*
id	bigint(10)	ID der Tabelle (Primärschlüssel)
mumiemodule	bigint(10)	ID der MUMIE-Aktivität
userid	bigint(10)	ID des Nutzers
groupid	bigint(10)	ID der Gruppe, in welcher der Nutzer im Kurs teilnimmt
grade	float(10, 2)	aktuelle Punktzahl des Nutzers innerhalb der Aktivität
timemodified	bigint(10)	Zeitpunkt der letzten Änderung (UNIX-Format)

Tabelle 3: mumiemodule_students – Informationen zu den einzelnen Studenten der Aktivität

Die erste Tabelle `mumiemodule` beinhaltet die Informationen zu jeweiligen Instanzen einer MUMIE Aktivität. Hierbei werden der Name und die Beschreibung aus dem entsprechenden Formular übernommen. Die Inhalte zu den Spalten ID, Lehrveranstaltungs-ID, und Zeitpunkt der letzten Änderung werden automatisch zusammengesucht und eingetragen. Das Feld mit den Informationen über die in dieser Aktivität maximal erreichbaren Punkte wird beim Erstellen der Instanz per HTTP-GET aus MUMIE geholt. Da sich im Laufe einer Lehrveranstaltung etwas daran ändern kann, wird diese Information bei jedem Zugriff auf die Aktivität in Moodle auf die selbe Art aktualisiert.

Die zweite Tabelle `mumiemodule_students` beschreibt die Teilnehmer einer einzelnen Aktivität und deren bereits erbrachte Leistungen näher. Für jeden Teilnehmer einer Instanz wird eine neue Zeile generiert, in welcher seine Nutzer-ID, die ID der Gruppe, in welcher er sich befindet, und sein aktueller Punktestand festgehalten wird. Auch hier wird bei jedem Zugriff auf diese Informationen in Moodle eine Liste der aktuellen Punktzahlen der Teilnehmer per HTTP-GET aus der MUMIE geholt und die entsprechenden Zeilen aktualisiert. So bleibt gewährleistet, dass die für den Nutzer dargestellten Informationen immer auf dem neuesten Stand sind.

Für beide Tabellen gilt, dass bei jeder Änderung einer Zeile, das Feld timemodified mit dem entsprechenden Zeitstempel im UNIX Format versehen wird.

4.2. Aufbau der MUMIE Aktivität in Moodle

Nachdem in Abschnitt 2.3.1.3. bereits erläutert wurde, was eine Aktivität in Moodle im semantischen Sinne ist und welche didaktischen Möglichkeiten sie im Rahmen einer Lehrveranstaltung erfüllt, wird nun der syntaktische Aspekt beleuchtet werden. Dabei werden einleitend allgemeine Gesichtspunkte von Aktivitäten betrachtet werden. Damit sind vor allem Vorgaben in der Verzeichnisstruktur und den Namenskonventionen gemeint, die notwendig sind, um den Datenfluss zwischen dem Aktivitätenmodul und dem Kern-Code von Moodle zu gewährleisten.

Im Anschluss daran wird der Kern dieser Arbeit – die Implementierung der MUMIE Aktivität – mit all seinen Besonderheiten im Blickfeld stehen. Dabei wird unterschieden werden zwischen der Kommunikation, die innerhalb von Moodle stattfindet und jener, die mit Hilfe des JapsClients Informationen mit MUMIE synchronisiert.

4.2.1. Allgemeine Vorgaben

Damit die Funktionalitäten eines Aktivitätenmoduls korrekt gewährleistet sein können, müssen im Aufbau der Ordnerstruktur und der darin enthaltenen Skripte bestimmte Konventionen eingehalten werden. Die Aktivitätenmodule befinden sich unter dem Ordner `moodle/mod`. Jedes Modul ist hier unter einem eigenen Unterordner zu finden. Dieser ist betitelt mit dem Namen der Aktivität in Kleinschreibung und beinhaltet die folgenden Pflichtelemente:

- `mod_form.php`: Dieses Skript enthält eine Klasse, welcher der Formularbibliothek von Moodle entspricht. Hiermit wird dasjenige Formular beschrieben, welches beim Erstellen oder Bearbeiten einer Instanz der jeweiligen Aktivität angezeigt wird.
- `version.php`: Diese Datei enthält Metainformationen bezüglich der vorliegenden Version des Moduls.
- `icon.gif`: Ein 16 x 16 Pixel großes Symbol für das Aktivitätenmodul.
- `index.php`: Diese Datei dient zum Anzeigen aller Instanzen der jeweiligen Aktivität.
- `view.php`: Diese Datei dient zum Anzeigen einer Instanz der jeweiligen Aktivität.

- `config.html`: Dies ist ein Formular zur Eingabe allgemeiner Konfigurationseinstellungen des Aktivitätenmoduls.
- `lib.php`: Diese Datei beinhaltet alle wichtigen Methoden, die für einen korrekten Ablauf nötig sind. Hierbei sind bestimmte Namenskonventionen für die vom Kern geforderten Methoden einzuhalten. Diese müssen den Namen der Aktivität in Kleinschreibung mit anschließendem Unterstrich als Prefix beinhalten(z. B. `mumiemodule_add_instance`), so dass die jeweilige Funktion vom Kern Code von Moodle gefunden werden kann.

  Die benötigten Methoden, die dort enthalten sein müssen sind unter anderen:

  - `modulename_add_instance()`: Diese Methode erstellt eine neue Instanz der jeweiligen Aktivität.
  - `modulename_update_instance()`: Diese Methode aktualisiert eine bestehende Instanz der jeweiligen Aktivität.
  - `modulename_delete_instance()`: Diese Methode löscht eine bestehende Instanz der jeweiligen Aktivität.

  Zusätzlich zu den vom Kern geforderten Methoden sollten in dieser Datei auch alle anderen Methoden beinhalten, die zum Ablauf der Aktivität nötig sind, sofern diese nicht in spezielle andere Dateien ausgelagert wurden.

Des weiteren müssen sich im Verzeichnis des jeweiligen Aktivitätenmoduls noch die beiden Unterverzeichnisse `/db` und `/lang` befinden. In `/lang` sollten die Sprachdateien mit den entsprechenden Strings für die jeweils installierten Sprachpakete zu finden sein.

Der Ordner `/db` beinhaltet:

- `install.xml`: Diese Datei beschreibt die neuen Teiltabellen des Aktivitätenmoduls für die Moodle Datenbank.
- `access.php`: Diese Datei definiert Rechte zu bestimmten Fähigkeiten innerhalb der jeweiligen Aktivität.
- `events.php`: Für die Anwendung der Events-API, die eine Neuerung seit Moodle 1.9 darstellt und in Abschnitt 4.2.3. genauer erläutert wird, ist diese Datei nötig. Innerhalb dieser wird spezifiziert, welche Module bei bestimmten Events informiert werden sollen und wo die entsprechenden Methoden zu finden sind.

4.2.2. Zusätzliche Elemente

Neben den Pflichtelementen eines Aktivitätenmoduls in Moodle wurden noch zusätzliche Ordner und Dateien erstellt, um die gewünschten Funktionalitäten der vorliegenden MUMIE Aktivität zu gewährleisten. Diese wurden so strukturiert, dass sich diejenigen Dateien, welche Methoden zur internen Kommunikation enthalten im Wurzelverzeichnis des Aktivitätenmoduls befinden, während die Methoden zur externen Kommunikation mit MUMIE in dem Unterverzeichnis `/japs_sync` definiert sind.

Die zusätzlichen Dateien zur internen Kommunikation in Moodle:

- `internalchecks.php`: Diese Datei enthält Methoden zur Überprüfung, ob bestimmte (Pseudo-)Objekte aus Moodle bereits in MUMIE bekannt sind. So können überflüssige Synchronisationsanfragen an MUMIE eingespart werden. Auf die einzelnen Abfragen wird im folgenden Abschnitt genauer eingegangen werden.
- `handler.php`: Diese Datei beinhaltet Methoden, die automatisch aufgerufen werden, wenn bestimmte Ereignisse (sog. Events) innerhalb des Moodle Kerns auftreten. Dieses Eventkonzept und die entsprechenden Methoden sind eine Neuerung seit Moodle 1.9 und werden ebenfalls im folgenden Abschnitt genauer beleuchtet werden.

Die Dateien zur externen Kommunikation mit MUMIE im Unterverzeichnis `/japs_sync`:

- `transfer_objects.php`: Diese Datei enthält die Klassendefinitionen der für MUMIE relevanten (Pseudo-)Objekte Semester, Lehrveranstaltung, Tutorium und Nutzer, wobei zusätzlich die Klassen Dozent, Tutor und Student definiert werden, welche Kindelemente von Nutzer sind.
- `hooks.php`: Diese Datei enthält neben einigen Hilfsfunktionen diejenigen Methoden für Moodle, welche im großen und ganzen die Synchronisationsbefehle zur MUMIE implementieren. Diese machen, wie bereits in Abschnitt 3.3.3. erläutert, Gebrauch von der Klasse JapsClient, welche dazu dient, dass sich das LMS als HTTP-Client bei MUMIE anmelden kann.

- `JapsClient.class.php`: Diese Datei enthält eine Klassendefinition des JapsClient in PHP. Innerhalb dieser Klasse befinden sich die Klassenvariablen, Methodendefinitionen zum Login oder zum Umgang mit Cookies. Diese Klasse bildet die Grundlage für die Authentifizierung von Moodle als Client bei MUMIE.
- `JapsSynchronise.class.php`: Innerhalb dieser Datei wird eine Klasse JapsSynchronise definiert, die von JapsClient erbt und zusätzliche Methoden zum Setzen von Name und Passwort bereitstellt. Mit diesen Angaben, die zur Laufzeit aus der Moodle Datenbank geholt werden, wird das LMS letztlich als Client bei MUMIE identifiziert.
- `LoginDialogResult.class.php`: Diese Datei definiert eine Klasse, die ein zusammengehörendes Paar von einem Nutzernamen als String und einem Passwort als Array repräsentiert.
- `HttpURLConnection.class.php`: Diese Datei definiert die Java-Klasse HttpURLConnection in PHP.

4.2.3. Interne Kommunikation in Moodle

Wie bereits erwähnt wurde, müssen bestimmte Namenskonventionen für die Ordner-, Datei- und Methodennamen eines Aktivitätenmoduls eingehalten werden, damit dessen Funktionalität gewährleistet ist. Durch das Befolgen dieser Konventionen kann sichergestellt werden, dass an den entsprechenden Stellen im Moodle Kern Code das Auffinden der jeweiligen Methoden fehlerfrei funktioniert. Dort wird dann eine Liste aller Module aus der Datenbank geholt, für jedes innerhalb der aktuellen Lehrveranstaltung instanzierte Modul die entsprechende Datei aus dem Verzeichnis des Moduls eingebunden und dann die benötigte Methode ausgeführt. Auf diese Weise findet Moodle automatisch den richtigen Weg, um eine Aktivität zu erstellen, zu aktualisieren oder zu löschen. Auf diese Weise kann ein Aktivitätenmodul über nahezu alle Ereignisse, die für den reibungslosen Ablauf der Aktivität von Relevanz sein können, informiert werden.

Dies bezieht sich allerdings nur auf Aktivitäten, deren Informationen einzig in der Datenbank von Moodle gespeichert werden. Für ein Modul wie das der hier behandelten MUMIE Aktivität, welche bestimmte Daten über mehrere Datenbanken konsistent halten muss, fehlt an einigen wichtigen Stellen

eine Möglichkeit, den Code des Moduls über Änderungen in Kenntnis zu setzen. So gibt es beispielsweise keine Möglichkeit, ein Aktivitätenmodul über neue oder Änderungen an vorhandenen Gruppen zu informieren. Für rein auf Moodle Seite agierende Module ist dies auch nicht nötig, da die entsprechenden Informationen zu den Gruppen innerhalb einer Lehrveranstaltung durch den Kern Code über die interne Datenbank ermittelbar sind. Nach den Anforderungen, die der Entwicklung des hier behandelten Aktivitätenmoduls zugrunde liegen, ist es allerdings nötig, dass diese Aktivität etwa über die Erstellung einer neuen Gruppe innerhalb einer Lehrveranstaltung, für welche bereits eine Instanz der Aktivität erstellt wurde, informiert wird. Weitere wichtige Ereignisse für den Ablauf der MUMIE Aktivität sind jegliche Änderungen an Gruppen und deren Zusammensetzungen, da die Zuordnung von Studenten zu Lehrveranstaltungen innerhalb von MUMIE nur über die Tutorien, welche ja in Moodle durch Gruppen repräsentiert werden, vorgenommen wird. Weiter müssen jegliche Änderungen von Daten über Nutzer, welche an MUMIE Aktivitäten teilnehmen, sofort an MUMIE gesendet werden, damit die Datensätze konsistent bleiben. Ebenso müssen Moodle Kurse, in welchen eine MUMIE Aktivität existiert, Änderungen im Namen oder der Beschreibung des Kursen an MUMIE schicken, da auch diese Informationen zwischen den beiden Plattformen synchronisiert werden.

Es fehlten also für die geforderte Funktionalität des zu implementierenden Aktivitätenmoduls einige fundamentale Aufrufe. Für die vorliegende Arbeit stellte dies eine besondere Schwierigkeit dar, weil es nun keine Möglichkeit mehr gab, die gewünschten Funktionen ohne Erweiterungen am Kern in Moodle zu integrieren. Jegliche Änderungen am Kern Code würden aber wiederum selbst gegen die Anforderungen verstoßen, weil dann ein reibungsloses Update von Moodle nicht mehr gewährleistet sein konnte. Nach einem solchen Update bestünde durchaus die Gefahr, dass eben jene Änderungen, die für das vorliegende Modul nötig waren, wieder entfernt oder überschrieben wurden. Die Lösung für dieses Dilemma lag in der Entwicklung der Moodle Event-API.

Die Event-API ist eine Neuerung für Moodle 1.9, welche sich zur Zeit noch in der Entwicklung befindet. Das Ziel dieser Entwicklung ist es, eine Kommunikationsmöglichkeit zwischen Modulen zu realisieren, so dass neue Module bei der Installation anmelden, über welche Ereignisse sie informiert werden und welche Methoden als Folge ausgeführt werden sollen. Die Informationen zu diesen Events und den dazugehörigen Handler-Methoden werden in spezielle Tabellen innerhalb der Moodle Datenbank geschrieben. Wird nun im Kern Code ein Event gefeuert, so wird innerhalb dieser Tabellen geschaut, ob für dieses Ereignis Handler registriert wurden und diese gegebenenfalls nacheinander abgearbeitet. Unter der Voraussetzung, dass diese Event-API funktioniert und an den entsprechenden Stellen im Moodle Kern die nötigen Ereignisse gefeuert werden, wäre es nun möglich, innerhalb des

Aktivitätenmoduls die passenden Handler-Methoden zu definieren und anzumelden.

Da aber wie erwähnt dieses Modell noch in der Entwicklung steckt und auch die nötigen Events noch nicht im Kern Code standen, war es vor der Fortführung der Arbeit an der MUMIE Aktivität nötig, diese Event-API soweit voranzutreiben, dass sie als zusätzliche Basis für die Weiterarbeit an der MUMIE Aktivität dienen konnte. Um dies zu erreichen, wurde ein Request im Tracker für Moodle erstellt[8] und die nötigen Codefragmente mit Angabe ihrer benötigten Positionen innerhalb des Kern Codes entwickelt und dort veröffentlicht. Erst nachdem die verantwortlichen Entwickler den Code geprüft und in den Moodle Kern übernommen hatten, war der Weg für die weitere Entwicklung der MUMIE Aktivität geebnet.

Nachdem nun gewährleistet ist, dass die Aktivität bei Änderungen zu allen für sie relevanten Informationen informiert wird, ist der nächste Punkt, diese Informationen entsprechend an MUMIE zu senden. Diese externe Kommunikation, die im nächsten Abschnitt genau beschrieben wird, sendet prinzipiell für jedes neue oder jede Änderung an einem der für die Synchronisation wichtigen (Pseudo-) Objekte einen HTTP-Request mit dem entsprechenden Synchronisationsbefehl an MUMIE. Dies kann in einigen Fällen aber zur Folge haben, dass mehr Requests übertragen werden als nötig, da unter Umständen Objekte neu an MUMIE geschickt werden, die dort bereits existieren, was zu unnötigen Fehlern führen kann. Um dies zu vermeiden, wurden interne Abfragen implementiert, die vor dem jeweiligen Senden eines Requests prüfen, ob dieser auf Seiten von MUMIE überhaupt notwendig ist. Zwar führt dies zu Einbußen in der Performance der Synchronisation; jedoch sind diese Einbußen geringer als jene, die durch überflüssige Requests auftreten würden.

So ist es selbstverständlich nicht unüblich, innerhalb eines Semesters mehr als lediglich eine Lehrveranstaltung anzubieten. Daher wurde eine Funktion implementiert, die beim Erstellen einer neuen Aktivität – denn nur an dieser Stelle kann der Synchronisationsbefehl für ein neues Semester veranlasst werden – prüft, ob innerhalb der Moodle Kurskategorie, die ja ein Semester repräsentiert, bereits eine andere Lehrveranstaltung existiert, welche eine MUMIE Aktivität beinhaltet. In diesem Fall ist das Semester schon bekannt und der Request kann ausbleiben.

Der Fall, dass Nutzer, die in der Datenbank von MUMIE bereits existieren, ein weiteres Mal geschickt werden, kann recht häufig eintreten:

- Neue Lehrer können MUMIE bekannt gemacht werden, wenn eine neue Aktivität angelegt wird oder wenn sie neu zu einem Kurs hinzugefügt werden, der bereits über eine MUMIE Aktivität verfügt. In beiden Fällen kann es sein, dass der Lehrer als Nutzer bereits in MUMIE steht, weil er

8 [MoT oJ]

etwa Kursleiter eines anderen Kurses ist. Daher wird an den entsprechenden Stellen im Code für alle Kurse, welche bereits über solche Aktivitäten verfügen, geprüft, ob der Nutzer dort bereits als Dozent eingetragen ist. Wenn dem so ist, steht er bereits als Nutzer in der Datenbank von MUMIE und muss nicht ein weiteres Mal per Synchronisationsbefehl bekannt gemacht werden. Theoretisch wäre es auch möglich, dass er mit der Rolle Tutor oder Student in der Datenbank steht. Allerdings ist dies in der Handhabung von MUMIE sehr unwahrscheinlich, weshalb diese Fälle vorerst nicht berücksichtigt wurden.

- Tutoren können ebenfalls beim Anlegen einer neuen Aktivität oder beim Einfügen eines Nutzers mit entsprechenden Berechtigungen in eine Gruppe an MUMIE geschickt werden. Die Abfrage an den jeweiligen Stellen geht ähnlich vor sich wie bei den Dozenten. So werden aus jedem Kurs, in welchem sich bereits eine MUMIE Aktivität befindet, die Gruppen dahingehend geprüft, ob der Nutzer bereits irgendwo als Tutor eingetragen ist. Wenn dem so ist, muss er nicht neu geschickt werden. Im Gegensatz zum oben beschriebenen Fall ist es hier aber nötig, sicherheitshalber auch zu prüfen, ob der neue Tutor evtl. in einem anderen Kurs als Dozent fungiert. Dafür wird die oben beschriebene Vorgehensweise genutzt. Nur wenn beides nicht der Fall ist, muss er MUMIE als neuer Nutzer bekannt gemacht werden.
- Studenten nehmen für gewöhnlich innerhalb ihres Studiums nicht bloß an einer einzigen Mathe-Veranstaltung teil. Deshalb ist die Wahrscheinlichkeit, dass sie in mehreren Lehrveranstaltungen (oder besser Gruppen) sind, recht hoch. Daher können mit einer geeigneten Abfrage hierzu wohl die meisten überflüssigen HTTP-Requests eingespart werden. Da Studenten nur über die Zugehörigkeit zu Tutorien in MUMIE an Lehrveranstaltungen gebunden werden können, sind die Stellen im Code, an denen dieser Sachverhalt geprüft werden muss, die selben wie bei den Tutoren. Lediglich das Prüfen selbst ist hier noch einfacher. Da nämlich alle Studenten, die an mindestens einer MUMIE Aktivität teilnehmen, in der entsprechenden Tabelle auftauchen, die in Abschnitt 4.1 beschrieben wurde, kann einfach geprüft werden, ob der einzufügende Student innerhalb dieser Tabelle schon existiert und MUMIE somit schon bekannt ist.

4.2.4. Externe Kommunikation mit MUMIE

Um die verschiedenen Informationen zwischen den beiden Lernplattformen synchron zu halten, ist es neben der oben beschriebenen internen Kommunikation unerlässlich, dass auch die externe Kommunikation mit MUMIE entsprechend funktioniert. Wie bereits mehrfach erwähnt wurde, macht Moodle dabei Gebrauch von der Klasse JapsClient, so dass sich das LMS als Client über HTTP bei MUMIE authentifizieren und die Synchronisation anstoßen kann.

An den relevanten Stellen im Code des hier betrachteten Aktivitätenmoduls – also beim Anlegen von neuen die Synchronisation betreffenden Objekten bzw. bei Änderungen an vorhandenen – werden jeweils die entsprechenden Informationen aus der Moodle Datenbank ermittelt. Anschließend wird zu jedem (Pseudo-)Objekt durch die bereits beschriebenen internen Mechanismen geprüft, ob dieses bereits MUMIE bekannt oder ob eine Synchronisation notwendig ist. Wenn dem so ist, werden die ermittelten Informationen den Parametern der Synchronisationsbefehle entsprechend auf einem Objekt gespeichert. Weiter wird eine Instanz der Klasse JapsSynchronise erstellt, welche den Nutzernamen und das Passwort enthält, mit dem sich Moodle bei MUMIE authentifiziert. Auf dieser Instanz wird nun die POST Methode ausgeführt. Dieser wird zusammen mit dem passenden Synchronisationsbefehl inklusive dem dazugehörigen Pfad ein Array übergeben, welches die jeweiligen Parameter des Befehls beinhaltet. Diese Methode erledigt die Authentifizierung bei MUMIE, wenn dies nicht vorher schon passiert ist und sendet die Synchronisationsanfrage per HTTP-POST. Die Synchronisation der Daten selbst wird nun auf Seiten von MUMIE ausgeführt. Als Antwort auf diese Anfrage sendet MUMIE ein "OK" oder eine entsprechende Fehlermeldung als Inhalt des HTTP-Response. Dieser wird von Moodle ausgewertet und in ein spezifisches die Synchronisation betreffendes Logfile geschrieben. Sollte bei der Synchronisation ein Fehler auftauchen, so kann dieser mit dem Request, bei welchem er auftrat, dort ermittelt werden. Eine von der Art des Fehlers abhängige Ausnahmebehandlung wurde noch nicht implementiert, da im Laufe dieser Arbeit nach Einbau der internen Abfragen, welche im vorigen Abschnitt beschrieben wurden, keine Fehler mehr auftauchten. Auf diese fehlende Funktionalität wird in Abschnitt 5 noch einmal eingegangen werden.

Bei der Reihenfolge der Synchronisationsbefehle muss auf die von MUMIE vorgeschriebenen Abhängigkeiten geachtet werden. So müssen die jeweiligen (Pseudo-)Objekte in folgender Reihenfolge synchronisiert werden: Zuerst muss das Semester existieren, in welchem sich die Lehrveranstaltung befindet. Anschließend müssen die Nutzer eingetragen werden, welche als Dozenten für diese Lehrveranstaltung fungieren, worauf die Lehrveranstaltung selbst folgt. Danach muss mit den Tutorien

zur Lehrveranstaltung ähnlich umgegangen werden. Zuerst müssen der Tutor existieren, dann das Tutorium selbst und dann erst können die Studenten eingefügt werden, die am Tutorium teilnehmen.

Würde diese Reihenfolge bei der Synchronisation nicht eingehalten, so erhielt Moodle entsprechende Fehlermeldungen als Response von MUMIE.

Die Ermittlung der in MUMIE vorhandenen Informationen zu Kursen und Punkten funktioniert ähnlich. Der Unterschied hierbei liegt darin, dass die aufgerufene Methode auf der Instanz der Klasse JapsSynchronise weder Parameter noch einen Synchronisationsbefehl übergeben bekommt. Lediglich anhand des Pfades, an den die Methode den GET-Request richtet, wird identifiziert, welche Informationen ermittelt werden sollen. Diese werden dann von MUMIE als XML im HTTP-Resonse an Moodle gesendet. Aus diesem XML werden die benötigten Informationen auf ein Array geschrieben und weiter verarbeitet.

4.3. Darstellungs- und Repräsentierungsprobleme

MUMIE obliegt im Vergleich zu Moodle durch die Architektur der Datenbank wesentlich stärkeren Einschränkungen wie die zu synchronisierenden Objekte zusammenhängen müssen. Die dadurch resultierenden Probleme und die jeweils implementierten Lösungen sollen in diesem Abschnitt erläutert werden.

Da in MUMIE die Zuordnung von Studenten zu einer Lehrveranstaltung nur über die Tutorien möglich ist, musste auch in Moodle eine geeignete Entsprechung gefunden werden. Denn für das LMS sind Gruppen keine Pflicht innerhalb von Kursen. Um aber den Anforderungen der MUMIE zu entsprechen, wird so vorgegangen, dass immer mindestens eine Gruppe innerhalb des Kurses, für welchen eine MUMIE Aktivität angelegt werden soll, existieren muss, so dass deren Mitglieder synchronisiert werden. Studenten, die zwar Teilnehmer des Kurses sind, aber keiner Gruppe zugehören, werden MUMIE nicht bekannt gemacht. Um nun sicher zu gehen, dass sich keine menschlichen Fehler in der Nutzung von Moodle für die Aktivität einschleichen, gab es zwei Möglichkeiten zur Auswahl. Zum einen wäre es möglich, vor dem Erstellen einer solchen Aktivität auf vorhandene Gruppen innerhalb des Kurses zu prüfen und eine Fehlermeldung, welche den Vorgang unterbindet, anzuzeigen, wenn noch keine Gruppen existieren. Die andere Möglichkeit sieht vor, dass

auch wenn noch keine Gruppe existieren sollte, die vorhandenen Informationen wie Semester und Lehrveranstaltung synchronisiert werden. Im Anschluss an diese Aktion allerdings und jedesmal wenn sich ein Nutzer mit entsprechenden Dozentenrechten auf die Ansicht der erstellten Aktivität bewegt, wird eine Meldung angezeigt, welche auf die fehlenden Gruppen hinweist. Diese besagt, dass das Erstellen von Gruppen und die Zuweisung der Studenten für die Nutzung von MUMIE unabdingbar ist. Da durch diese Lösung der Dozent weniger durch Fehlermeldungen und Abbrüche eingeschränkt wird und diese Lösung auch für den Fall hilfreich ist, dass die letzte Gruppe im Nachhinein gelöscht werden sollte, wurde dieser Lösungsweg gewählt.

Gruppen werden innerhalb einer Lehrveranstaltung in Moodle insgesamt wesentlich freier gehandhabt als in MUMIE. Dadurch ist es hier auch möglich, dass Studenten in keiner oder auch mehreren Gruppen gleichzeitig teilnehmen können. MUMIE hingegen verlangt, dass ein Student in genau einem Tutorium pro Kurs sein muss.

Würde nun einfach die Darstellung in Moodle belassen werden und die Gruppenzugehörigkeiten einfach an MUMIE geschickt, so wäre für den Nutzer am Rechner nicht mehr zu erkennen, in welchem Tutorium in MUMIE der Student nun wirklich eingetragen wird. Als Lösung für dieses Problem blieb vorerst nichts anderes übrig, als Moodle ein wenig einzuschränken. Möchte ein Dozent nun eine MUMIE Aktivität erstellen, wo wird erst geprüft, ob es Studenten innerhalb des Kurses, in dem die Aktivität erstellt werden soll, gibt, die in mehreren Gruppen teilnehmen. Wenn dem so ist, erscheint eine entsprechende Fehlermeldung und der Vorgang wird nicht durchgeführt. Dies funktioniert erst, nachdem diese Situation vom Dozenten von Hand behoben wurde. Man könnte auch entsprechende Studenten automatisch aus bestimmten Gruppen entfernen. Allerdings ist die Frage, wie entschieden werden sollte, aus welchen. Daher wird dies dem Dozenten überlassen. Wenn bereits eine MUMIE Aktivität innerhalb eines Kurses existiert und ein Student nachträglich in eine Gruppe geschoben wird, so wird geprüft, ob dieser Student bereits in einer anderen Gruppe teilnimmt. Wenn dem so ist, wird der Student aus der neuen Gruppe automatisch wieder entfernt und eine entsprechende Fehlermeldung erscheint, welche den aktuellen Nutzer auf das Problem hinweist. Wenn er wirklich den Studenten in die neue Gruppe einfügen möchte, so muss er ihn erst aus der anderen Gruppe entfernen. In der Moodle Roadmap steht, für eine der nächsten Versionen sei die Funktionalität geplant, Aktivitäten bestimmten Gruppen zuweisen zu können. Somit bestünde die gerade beschriebene Problematik nur für solche Gruppen und es würden wahrscheinlich leichte Anpassungen am Code nötig werden.

Eine ähnliche Schwierigkeit stellt die Zuweisung von Nutzern mit Tutorenrechten zu Gruppen in Moodle dar. Während Studenten pro Lehrveranstaltung nicht an mehr als einem Tutorium teilnehmen dürfen, gilt für Tutoren in MUMIE, dass diese zwar mehrere Gruppen pro Kurs unterrichten dürfen; aber es muss pro Tutorium immer genau ein Tutor vorhanden sein. Durch die Tatsache, dass diese Vorgabe bereits beim Erstellen eines Tutoriums erfüllt sein muss, bekommt dieses Problem eine zusätzliche Schwere. In Moodle werden Gruppen nämlich grundsätzlich erst einmal ohne einen Tutor oder ähnliches erstellt. Wenn überhaupt werden diese als Mitglieder mit bestimmten Rechten innerhalb des umschließenden Kurses in die Gruppe eingefügt. Hierbei ist es natürlich in Moodle auch möglich, mehrere solcher Nutzer innerhalb einer Gruppe zu haben. Grundsätzlich sollte hier eine Änderung an der Datenbank von MUMIE vorgenommen werden. Genau wie dies bei der Handhabung von Lehrveranstaltungen der Fall ist, sollte hier eine weitere Tabelle eingefügt werden, in welcher die zugehörigen Tutoren zu einem Tutorium erfasst werden. Hierdurch würde sich die Möglichkeit ergeben, ebenfalls Tutorien ohne oder gar mit mehreren Tutoren zu haben. Dies wäre nicht bloß für die Synchronisation mit Moodle eine Erleichterung. Da viele LMS Gruppen ähnlich wie Moodle regeln, wäre es ein genereller Vorteil für die Synchronisation von MUMIE mit einem Master-LMS. Da nun eine solche zusätzliche Tabelle noch nicht existiert, musste vorerst noch eine andere Lösung für diese Problematik gefunden werden. Hierfür wurde diese in zwei Teilprobleme untergliedert. Als erstes wird der Fall betrachtet, dass ein Tutorium für MUMIE nicht mehr als einen Tutor haben darf. Diese Problematik wurde ähnlich gelöst wie das oben beschriebene Problem der Zuweisung von Studenten zu mehreren Gruppen. Wenn ein Dozent eine MUMIE Aktivität erstellen möchte, wird erst geprüft, ob es innerhalb des Kurses Gruppen gibt, welche mehr als einen Nutzer mit entsprechenden Tutorenrechten beinhalten. Wenn dem so ist, wird eine Fehlermeldung angezeigt, welche den Nutzer auf die Situation hinweist und der Vorgang wird abgebrochen. Existiert eine solche Aktivität bereits und ein Tutor wurde zu einer Gruppe hinzugefügt, so wird geprüft, ob nun in dieser Gruppe mehr als ein Nutzer mit Tutorenrechten existiert. In diesem Fall wird der gerade eingefügte Nutzer wieder entfernt und eine entsprechende Fehlermeldung angezeigt.

Das zweite Teilproblem, dass ein Tutorium für MUMIE immer schon einen Tutor beinhalten muss, tritt mindestens dann auf, wenn in einem Kurs, welcher bereits eine MUMIE Aktivitäten beinhaltet, nachträglich eine Gruppe angelegt wird. Denn wie bereits erwähnt fehlt an dieser Stelle der entsprechende Nutzer für MUMIE. Hierfür gab es mehrere Überlegungen, wie man das lösen könnte. So könnte man beispielsweise immer den ersten Dozenten des Kurses eintragen, was aber problematisch werden würde, wenn der Kurs noch keine Dozenten hat oder wenn gar nachträglich noch ein Dozent hinzugefügt wird. Zum anderen könnte man den aktuell angemeldeten Nutzer in

Moodle als Tutoren eintragen. Da dieser nämlich nur der Kursersteller oder gar der Dozent selbst sein kann, verfügen diese auch über die nötigen Rechte. Allerdings löst das nicht die Problematik der Darstellung. Denn in MUMIE wäre ein reeller Tutor eingetragen, wobei in Moodle ein solcher bei der Darstellung der Gruppen nicht zu erkennen wäre.

Die folgende Lösung, die für dieses Problem implementiert wurde, kann lediglich als Workaround betrachtet werden, bis die bereits erwähnte zusätzliche Tabelle, welche Tutoren zu Tutorien zuordnet, in die Datenbank von MUMIE eingebaut wurde. So wird momentan in der MUMIE Datenbank ein Dummy-Tutor erstellt. Dieser Nutzer wird jedesmal als Tutor einer Gruppe an die MUMIE geschickt, wenn eine MUMIE Aktivität erstellt wird in einem Kurs mit Gruppen ohne entsprechende Tutor-Nutzer, wenn eine neue Gruppe in einem Kurs mit existierender MUMIE Aktivität erstellt wird oder wenn in einem solchen Kurs ein Tutor aus einer Gruppe entfernt wird. So ist eine geeignete Darstellung noch am ehesten gewährleistet. Denn wenn dieser Nutzer als Tutor für ein Tutorium in der MUMIE Datenbank erscheint, ist sofort klar, dass in Moodle noch kein entsprechender Nutzer zur Gruppe hinzugefügt wurde. Auch sind die entsprechenden Stellen am Code des Aktivitätenmoduls ohne Schwierigkeiten wieder zu entfernen, wenn die gewünschten Änderungen in die Datenbank von MUMIE eingefügt wurden.

Eine weiter Schwierigkeit ist der Umgang mit mehreren MUMIE Aktivitäten pro Lehrveranstaltung, was in Moodle generell möglich ist. Zwar ist es im Prinzip auch zu realisieren, dass in MUMIE mehrere Kurse als Lerninhalte mit einer Lehrveranstaltung verbunden sind. Da dies jedoch in der Praxis nicht gewollt ist, wurde diese Möglichkeit in der vorliegenden Realisierung des MUMIE Aktivitätenmoduls durch eine Abfrage unterbunden. Wenn nun ein Nutzer in Moodle versucht, eine weiter Aktivität innerhalb einer Lehrveranstaltung, welche bereits über eine solche Instanz verfügt, zu erstellen, so erscheint eine Fehlermeldung, welche den Nutzer auf die Einschränkung hinweist und die weiteren Schritte werden abgebrochen.

All die hier beschriebenen Lösungen schränken die Funktionalitäten vor allem von Moodle ein. Daher ist es wünschenswert, dass die erwähnten Änderungen und Erweiterungen an der Datenbank von MUMIE vorgenommen werden. Im Anschluss sollten diese Einschränkungen aus dem Moodle Code wieder entfernt werden. Da aber Änderungen an MUMIE im Rahmen dieser Arbeit nicht gewünscht sind, muss so lange mit diesen Lösungen gearbeitet werden.

5. Ausblick

Neben den in Abschnitt 4.3. beschriebenen Darstellungsproblemen gibt es auch noch offene Fragen und Probleme, die im Rahmen der vorliegenden Bachelorarbeit nicht gelöst werden konnten. So sollte Moodle als Eingangsportal zur MUMIE fungieren, so dass Nutzer, die über das LMS in MUMIE eingetragen wurden, über einen Link zu MUMIE navigieren können. Hierzu müsste das Passwort und der Nutzernamen dynamisch aus der Moodle Datenbank ermittelt und damit die Authentifizierung bei MUMIE durchgeführt werden. Da aber das Passwort in der Datenbank von Moodle lediglich als MD5 Prüfsumme vorliegt, aus welcher sich kein Klartext mehr ermitteln lässt, war es nicht ohne weiteres möglich, diese Authentifizierung durchzuführen. Der Authentifizierungsmechanismus von MUMIE erwartet aber ein Passwort als Klartext. Aus diesem wird dann intern die MD5 Prüfsumme gebildet und mit dem Eintrag in der Datenbank verglichen. Die Lösung, die für das vorliegende Aktivitätenmodul implementiert wurde, sieht so aus, dass das bereits als Prüfsumme in Moodle vorliegende Passwort erneut gehasht wird und diese neue Prüfsumme dann als verschlüsseltes Passwort beim Eintragen eines neuen Nutzers an MUMIE gesendet wird. Navigiert der Nutzer nun über den Link zu Moodle, so wird die normale MD5 Prüfsumme aus Moodle ermittelt und zusammen mit dem Nutzernamen an MUMIE geschickt. Dort wird wird diese neu gehasht und stimmt nun mit dem Eintrag in der MUMIE Datenbank überein. So ist ein Workaround geschaffen, mit dem die geforderte Portalfunktion von Moodle gewährleistet ist. Allerdings ist es dem Nutzer nun nicht mehr möglich, sich direkt auf der Login-Seite von MUMIE zu authentifizieren, da er im Prinzip sein dortiges Passwort nicht kennt. Als Lösung für dieses Problem gäbe es die Möglichkeit, den Authentifizierungsmechanismus auf Seiten von MUMIE dahingehend zu erweitern, dass dieser zuerst das erhaltene Passwort als Klartext interpretiert, daraus die Prüfsumme bildet und diese mit dem Eintrag in der Datenbank vergleicht. Sollte hierbei keine Übereinstimmung gefunden werden, so müsste der Mechanismus diese Prüfsumme erneut hashen und wieder mit der Datenbank vergleichen. Da allerdings auch diese Lösung lediglich einem Workaround gleichkommt, wäre die sauberste Möglichkeit, beide Systeme mit einem gängigen Single-Sign-On Verfahren zu koppeln. Damit wäre ein angemeldeter Nutzer automatisch auf allen zugehörigen Systemen authentifiziert.

Zusätzlich zu der Portalfunktion im Allgemeinen wäre es wünschenswert, dass sich der Nutzer über den erwähnten Link auf der Seite des LMS nicht bloß auf die Startseite von MUMIE navigieren könnte. Vielmehr sollte er gleich zu der entsprechenden Lerneinheit auf MUMIE geleitet werden. Hierzu wäre es aber nötig, innerhalb von Moodle auch diese Lerneinheit – als den MUMIE Kurs – eindeutig

identifizieren zu können. Um dies zu erreichen, könnte etwa die entsprechende ID von MUMIE, welche beim Eintragen einer neuen Instanz der Aktivität als Parameter des Synchronisationsbefehls zum Erstellen einer neuen Lehrveranstaltung gesendet wird, in Moodle gespeichert werden. Allerdings ändert sich diese ID auf Seiten von MUMIE, sobald nachträglich Änderungen an diesem Lerninhalt vorgenommen werden, was durchaus nicht unüblich ist. Somit wäre die ID in der Moodle Datenbank nicht mehr aktuell. Eine Lösung für dieses Problem bestünde in der Einführung von Synchronisations-IDs für diese Lerninhalte auf Seiten von MUMIE ähnlich wie auch bei den anderen die Synchronisation betreffenden (Pseudo-)Objekten. Würden diese IDs auch bei einer nachträglichen Änderung an diesem Lerninhalt beibehalten, so wäre eine eindeutige Identifizierung über diese ID zwischen den beiden Lernplattformen möglich. Da dies aber eine Änderung am Code von MUMIE bedeuten würde, wurde diese Lösung im Rahmen der vorliegenden Arbeit nicht implementiert.

Der Einbau dieser Synchronisations-ID in MUMIE würde noch einen weiteren Vorteil für Moodle bedeuten. Dann wäre es nämlich möglich, dass Moodle an geeigneter Stelle Anfragen über die Aufgabenblätter und die dazugehörigen Abgabefristen zu den jeweiligen Lerninhalten an MUMIE sendet. So könnten in Zukunft noch Möglichkeiten in Moodle integriert werden, die Studenten über bald auslaufende Fristen zu bestimmten Aufgaben zu informieren oder diese gar per Mausklick direkt erreichbar zu machen.

Eine andere Schwierigkeit, welche die Lerninhalte auf Seiten von MUMIE betrifft, ist deren Zuordnung zu Lehrveranstaltungen. Wenn etwa beim Anlegen einer neuen MUMIE Aktivität in Moodle aus der angebotenen Liste ein MUMIE Kurs ausgewählt wird, der dort bereits mit einer anderen Lehrveranstaltung verbunden ist, so wird diese alte Verbindung beim Senden des Synchronisationsbefehls zum Anlegen einer neuen Lehrveranstaltung überschrieben. Denn in MUMIE kann eine solche Lerneinheit nur mit jeweils einer Lehrveranstaltung verbunden werden. Möglichkeiten dieses Problem zu umgehen, könnten auf beiden Systemen implementiert werden. Die Informationen über die auf MUMIE vorhandenen Lerninhalte, die von Moodle an der entsprechenden Stelle als XML angefordert werden, beinhalten auch Angaben, ob und mit welcher Lehrveranstaltung diese jeweils verbunden sind. Es wäre also möglich, schon beim Erstellen der Auswahlliste in Moodle alle Inhalte zu entfernen, die bereits in einer Lehrveranstaltung genutzt werden. Eine andere Möglichkeit, welche den Nutzer, der die Aktivität anlegen möchte, weniger einschränkt, wäre es, eine Hinweismeldung anzuzeigen, wenn ein MUMIE Kurs mit der aktuellen Lehrveranstaltung verbunden werden soll, obwohl dieser bereits Teil einer anderen Lehrveranstaltung ist. Diese Hinweismeldung sollte die Erstellung der neuen Aktivitäteninstanz unterbrechen und den Nutzer kurz über die Folgen unterrichten. Die Entscheidung, ob die unterbrochene Aktion fortgesetzt oder die Angaben geändert

werden sollen, bleibt dann dem Nutzer selbst überlassen. Eine weitere Lösung wäre es, auf Seiten von MUMIE die Möglichkeit einzubauen, eine solche Lerneinheit mit mehr als bloß einer Lehrveranstaltung zu verbinden. So könnten etwa Lerneinheiten aus einem älteren Semester mit neuen Lehrveranstaltungen verknüpft werden, welche die gleichen Inhalte vermitteln sollen.

Um den vollständigen Umfang an administrativen Aufgaben übernehmen zu können, ist es nötig, dass auch eine Möglichkeit existiert, bestimmte Objekte wieder zu löschen. Entsprechende Synchronisationsbefehle sind aber auf MUMIE noch nicht implementiert, womit sie auch nicht in die vorliegende Arbeit einfließen konnten. Wird nun eine Gruppe, eine ganze Instanz einer MUMIE Aktivität oder ein anderes (Pseudo-)Objekt in Moodle entfernt, so werden die entsprechenden Repräsentierungen innerhalb der Moodle Datenbank zwar gelöscht – in MUMIE allerdings bleiben sie weiter bestehen. Erst wenn an entsprechender Stelle in der MUMIE Datenbank ein boolescher Wert in den jeweiligen Zeilen manuell geändert wird, gilt der Inhalt als gelöscht, was aber dazu führen kann, dass die Datenbank mit der Zeit sehr groß werden kann. Es wäre wünschenswert, wenn diese fehlende Funktionalität zum echten Löschen von Inhalten bald gemeinsam mit den entsprechenden Synchronisationsbefehlen in MUMIE implementiert wird. Dann könnten diese Aufrufe auch in den Code des Aktivitätenmoduls in Moodle übernommen werden.

Ein weiterer noch offener Punkt ist die Fehlerbehandlung innerhalb des neu entwickelten MUMIE Aktivitätenmoduls anhand des im HTTP-Response eingebetteten Texts. Momentan existieren im Quellcode lediglich Platzhalter für entsprechende Aktionen, da nach dem Einbau der internen Prüfmethoden zum Einsparen von überflüssigen Synchronisations-Requests in Moodle keine Fehler mehr aufgetaucht sind. Allerdings ist es denkbar, dass Fehler auftauchen, auf welche die Entwickler selbst keinen Einfluss nehmen können wie etwa Übertragungsfehler. Aus diesem Grund ist es für die Zukunft geplant, eine geeignete Fehlerbehandlung zu entwickeln. Hierbei wäre es beispielsweise denkbar, einen Fehlerstack zu schaffen, auf welchem Synchronisationsbefehle gespeichert werden, welche nicht fehlerfrei ausgeführt wurden. Diese könnten dann nach einer Wartezeit automatisch erneut ausgeführt werden, so dass solche Übertragungsfehler minimiert werden.

6. Fazit

Auch wenn gewisse Funktionalitäten noch nicht eingebaut oder lediglich durch einen Workaround realisiert wurden, so ist mit dem hier behandelten Aktivitätenmodul doch eine voll und ganz den Anforderungen entsprechende Implementierung der Synchronisation und Integration der beiden Lernplattformen MUMIE und Moodle geschaffen worden, welche das Arbeiten mit den beiden Systemen maßgeblich vereinfacht. Moodle fungiert dabei als vollwertiges Master-LMS, welches an den jeweils passenden Stellen die administrativen (Pseudo-)Objekte Semester, Lehrveranstaltung, Tutorien und Nutzer selbständig und automatisch für MUMIE anlegt oder aktualisiert. Hierfür sind keine redundanten Arbeitsschritte nötig. Alle Einstellungen und Zuweisungen - etwa von Studenten zu Gruppen - werden in Moodle vorgenommen und dann von selbst mit MUMIE synchronisiert. Die gewohnten Abläufe in Moodle haben sich dabei nicht verändert, so dass sich erfahrene Nutzer nicht umstellen müssen. Zwischen den beiden Lernplattformen fungiert Moodle als Eingangsportal zu MUMIE. Nutzer können direkt über das LMS auf die Startseite von MUMIE gelangen, ohne sich neu authentifizieren zu müssen. Es ist also kein Bruch zwischen den beiden Systemen wahrnehmbar. Die Darstellung der momentanen Gesamtpunktzahl der Studenten wurde dem Moodle Standard entsprechend eingebaut, wobei unterschieden wird zwischen der Ansicht für Studenten und derjenigen für Dozenten und Tutoren, die mehr Informationen dargestellt bekommen.

Gemäß den Anforderungen wurde die Integration als Aktivitätenmodul so modular implementiert, dass sie in eine laufende Moodle Instanz ohne Probleme eingefügt werden kann und auch Updates keine Schwierigkeiten darstellen. Die Erweiterungen, welche am Kern Code nötig waren, wurden ausreichend mit den Hauptentwicklern kommuniziert und werden von diesen in das kommende Moodle Release übernommen werden, so dass die in dieser Bachelorarbeit thematisierte MUMIE Aktivität ab Moodle 1.9 im vollen Umfang genutzt werden kann.

7. Quellen / URLs

[Ete 06] e-teaching.org, Prof. Dr. Dr. Friedrich W. Hesse, Institut für Wissensmedien, Tübingen http://www.e-teaching.org/technik/distribution/lernmanagementsysteme/ (Stand: 04. Januar 2008)

[LAM 02] LAMS International, http://www.lamsinternational.com/ (Stand: 28. November 2007)

[MoD 07] Moodle Docs, http://docs.moodle.org/de/Was_ist_moodle%3F (Stand: 09. Januar 2008)

[Moo 0J] Moodle, http://moodle.org/ (Stand: 09. Januar 2008)

[MoT 0J] Moodle Tracker, http://tracker.moodle.org/browse/MDL-9983 (Stand: 15. Oktober 2007)

[MUM 0J] MUMIE, Technische Universität Berlin, http://www.mumie.net/ (Stand: 04. Januar 2008)

[Sch 03] Schulmeister, R. (2003): Lernplattformen für das virtuelle Lernen, Evaluation und Didaktik. München: Oldenbourg.

8. Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

---------- ----------

Ort, Datum Unterschrift

9. Anhang

A1.1 Quellcode mod_form.php

```
<?php

global $CFG;
require_once $CFG->dirroot.'/course/moodleform_mod.php';
require_once 'internalchecks.php';

class mod_mumiemodule_mod_form extends moodleform_mod {

    function definition() {
        global $USER;
        global $CFG;
       global $COURSE;

       //business-rules for the activity
        if(!isset($CFG->syncServer) || !isset($CFG->syncUser) ||
                  !isset($CFG->syncPassword)){ //configuration settings
            error(get_string('nosettings', 'mumiemodule'));
        }
        //right now, we do not allow more than one activity per class - so:
        if(record_exists('mumiemodule', 'course', $COURSE->id)){
            error(get_string('justonemod', 'mumiemodule'),
                                        "view.php?id=".$COURSE->id);
            break;
        }
       /*if (!get_groups($COURSE->id)) { //at least one group
           error(get_string("nogroup", "mumiemodule"),
                           "../course/view.php?id=".$COURSE->id);
           }*/
        //groups are not allowed to have more than one tutor
        //(actually a group should have exactly one)
        $coursegroups = groups_get_groups($COURSE->id);
        $fields = 'u.id';
        foreach($coursegroups as $coursegroup){
            $groupcontext = get_context_instance(CONTEXT_GROUP, $coursegroup);
            $grouptutors = get_users_by_capability($groupcontext,
                                                   'mod/mumiemodule:tutorize',
                                                    $fields, '', '', '',
                                                    $coursegroup);
            if(count($grouptutors)>1){
                error(get_string('toomuchtutors', 'mumiemodule'),
                                    "../group/index.php?id=".$COURSE->id);
                break;
            }
        }
       //a student must not be in more than one group
        $coursecontext = get_context_instance(CONTEXT_COURSE, $COURSE->id);
        $coursestudents = get_users_by_capability($coursecontext,
                                                  'mod/mumiemodule:participate',
                                                     'u.id');
        foreach($coursestudents as $coursestudent){
            $studentsgroups = groups_get_groups_for_user($coursestudent->id,
                                                         $COURSE->id);
```

```
            if(count($studentsgroups)>1){
                error(get_string('justonegroup', 'mumiemodule'),
                        "../group/index.php?id=".$COURSE->id);
                break;
            }
        }

        $mform =& $this->_form;

        $mform->addElement('header', 'general', get_string('general', 'form'));

        //name, inputfield
        $mform->addElement('text', 'name', 'Name', array('size'=>'64'));
        $mform->setType('name', PARAM_TEXT);
        $mform->addRule('name', null, 'required', null, 'client');

       //description, textarea
        $mform->addElement('htmleditor', 'description',
                           get_string('description', 'mumiemodule'));
        $mform->setType('description', PARAM_RAW);
        $mform->setHelpButton('description', array('writing', 'questions',
                                'richtext'), false, 'editorhelpbutton');

       //mumie-courses, combobox
       $mumie_courses = get_mumie_courses();
        $selArray = array();
        //just for later checks:
        $alreadyusedcourses = '';

       foreach ($mumie_courses as $mumie_course) {
            $selArray[$mumie_course->id]=$mumie_course->name;

            //for having a string-list of courses, that are already used
            if($mumie_course->alreadyused===true){
                $alreadyusedcourses.=$mumie_course->id.' ';
            }
        }
       $mform->addElement('select', 'mumie_course_id', get_string('mumiecourse',
                           'mumiemodule'), $selArray);
        //small workaround to give the IDs of the courses already used
        //to our add_instance function:
        $mform->addElement('hidden', 'mumie_courses_alreadyused',
                           $alreadyusedcourses);

       //common module settings
        $this->standard_coursemodule_elements();

       //action buttons
        $this->add_action_buttons();
    }
}
?>
```

A1.2 Quellcode index.php

```
<?php // $Id: index.php,v 1.5 2006/08/28 16:41:20 mark-nielsen Exp $
/**
 * This page lists all the instances of mumiemodule in a particular course
 *
 * @author PR
 * @version $Id: index.php,v 1.5 2006/08/28 16:41:20 mark-nielsen Exp $
 * @package newmodule
 **/


    require_once("../../config.php");
    require_once("lib.php");

    $id = required_param('id', PARAM_INT);   // course

    if (! $course = get_record("course", "id", $id)) {
        error("Course ID is incorrect");
    }

    require_login($course->id);

    add_to_log($course->id, "mumiemodule", "view all",
                "index.php?id=$course->id", "");


/// Get all required stringsmumiemodule

    $strmumiemodules = get_string("modulenameplural", "mumiemodule");
    $strmumiemodule  = get_string("modulename", "mumiemodule");

    //neu von PR:
    $strgrade = get_string("grade");


/// Print the header

    if ($course->category) {
        $navigation = "<a href=\"../../course/view.php?id=$course->id\">
                        $course->shortname</a> ->";
    } else {
        $navigation = '';
    }

    print_header("$course->shortname: $strmumiemodules", "$course->fullname",
                "$navigation $strmumiemodules", "", "", true, "",
                 navmenu($course));

/// Get all the appropriate data

    if (! $mumiemodules = get_all_instances_in_course("mumiemodule", $course)) {
        notice("There are no mumiemodules",
                "../../course/view.php?id=$course->id");
        die;
    }
```

```
/// Print the list of instances (your module will probably extend this)

    $timenow = time();
    $strname  = get_string("name");
    $strweek  = get_string("week");
    $strtopic  = get_string("topic");

    if ($course->format == "weeks") {
        $table->head  = array ($strweek, $strname, $strgrade);
        $table->align = array ("center", "left");
    } else if ($course->format == "topics") {
        $table->head  = array ($strtopic, $strname, $strgrade);
        $table->align = array ("center", "left", "left", "left");
    } else {
        $table->head  = array ($strname, $strgrade);
        $table->align = array ("left", "left", "left");
    }

    foreach ($mumiemodules as $mumiemodule) {
        if (!$mumiemodule->visible) {
            //Show dimmed if the mod is hidden
            $link = "<a class=\"dimmed\"
                      href=\"view.php?id=$mumiemodule->coursemodule\">
                      $mumiemodule->name</a>";
        } else {
            //Show normal if the mod is visible
            $link = "<a href=\"view.php?
                        id=$mumiemodule->coursemodule\">$mumiemodule->name</a>";
        }

        if ($course->format == "weeks" or $course->format == "topics") {
            $table->data[] = array ($mumiemodule->section, $link);
        } else {
            $table->data[] = array ($link);
        }
    }

    echo "<br />";

    print_table($table);

/// Finish the page

    print_footer($course);

?>
```

A1.3 Quellcode view.php

```
<?php  // $Id: view.php,v 1.4 2006/08/28 16:41:20 mark-nielsen Exp $
/**
 * This page prints the current sudents points of a current instance of a
mumiemodule
 * and links to the MUMIE itself
 *
 * @author PR
 * @version $Id: view.php,v 1.4 2006/08/28 16:41:20 mark-nielsen Exp $
 **/


    require_once("../../config.php");
    require_once("lib.php");

    $id = optional_param('id', 0, PARAM_INT); // Course Module ID, or
    $a  = optional_param('a', 0, PARAM_INT);  // mumiemodule ID
    $changegroup = optional_param('group', -1, PARAM_INT);
      // choose the current group

    if ($id) {
        if (! $cm = get_record("course_modules", "id", $id)) {
            error("Course Module ID was incorrect");
        }

        if (! $course = get_record("course", "id", $cm->course)) {
            error("Course is misconfigured");
        }

        if (! $mumiemodule = get_record("mumiemodule", "id", $cm->instance)) {
            error("Course module is incorrect");
        }

    } else {
        if (! $mumiemodule = get_record("mumiemodule", "id", $a)) {
            error("Course module is incorrect");
        }
        if (! $course = get_record("course", "id", $mumiemodule->course)) {
            error("Course is misconfigured");
        }
        if (! $cm = get_coursemodule_from_instance("mumiemodule",
             $mumiemodule->id, $course->id)) {
            error("Course Module ID was incorrect");
        }
    }

    require_login($course->id);
    $context = get_context_instance(CONTEXT_MODULE, $cm->id);

    //get the current grades out of the MUMIE
    $mumie_grades = get_mumie_grades($course->id);
    $array_length = count($mumie_grades);
    $iterator = 0;
    foreach($mumie_grades as $mumie_grade){
        $iterator++;
```

```
        if($iterator<$array_length){
            $mumiemodule_student = new object();
            $mumiemodule_student->id = get_field('mumiemodule_students', 'id',
                                                 'mumiemodule',
                                                  $mumiemodule->id, 'userid',
                                                  $mumie_grade->userid);
            $mumiemodule_student->mumiemodule = $mumiemodule->id;
            $mumiemodule_student->userid = $mumie_grade->userid;
            $mumiemodule_student->groupid = get_field('mumiemodule_students',
                                                      'groupid', 'id',
                                                    $mumiemodule_student->id);
            $mumiemodule_student->grade = $mumie_grade->current_points;
            //$mumiemodule_student->lastexercise
            $mumiemodule_student->timemodified = time();
            update_record('mumiemodule_students', $mumiemodule_student);
        }else{
            $mumiemodule->grade = $mumie_grade->total_points;
            update_record('mumiemodule', $mumiemodule);
        }
    }

    add_to_log($course->id, "mumiemodule", "view", "view.php?id=$cm->id",
               "$mumiemodule->id");

/// Print the page header

    if ($course->category) {
        $navigation = "<a href=\"../../course/view.php?id=$course->id\">
                        $course->shortname</a> ->";
        $navigation .= "<a href=\"view.php?id=$id\">$mumiemodule->name</a> ->";
    } else { //when does this happen???
        $navigation = '';
    }

    $strmumiemodules = get_string("modulenameplural", "mumiemodule");
    $strmumiemodule  = get_string("modulename", "mumiemodule");

    print_header("$course->shortname: $mumiemodule->name->current student
                                            points", "$course->fullname",
                 "$navigation <a href=index.php?id=$course->id>
                                $strmumiemodules</a>",
                  "", "", true, ' ',
                  navmenu($course, $cm));
```

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////
/// Print the main part of the page
print_heading(get_string("modulename", "mumiemodule"));

//Link to enter the MUMIE:
$mumielink = "";
    print_simple_box_start('right');
    echo '<script type="text/javascript">function openmumie2()
            {document.mumieform2.target =
            "_blank";document.mumieform2.submit();}</script>'
        .'<form name="mumieform2" action="'.
            $CFG->syncServer.'/public/auth/login" method="post">'
        .'<input type="hidden" name="name" value="'.$USER->username.'" />' .
                '<input type="hidden" name="password" value="'
                  .$USER->password.'" />' .
                '<input type="hidden" name="resource" value="'
                  .$CFG->syncServer.'/protected/auth/login-successful" />
                  <a href="javascript:openmumie2()" > ' .
                get_string('entermumie', 'mumiemodule').'</a>'
        .'</form>';
    print_simple_box_end();
//end of entering
//show info
    if (!empty($mumiemodule->description)) {
                print_box(format_text($mumiemodule->description), 'generalbox',
                                    'description');
            }

if(has_capability('mod/mumiemodule:manage', $context)){
    $datetime =    date("F d,Y g:i a");

    /// Some capability checks.
    //shall we see hidden activities?
    if (empty($cm->visible)
        and !has_capability('moodle/course:viewhiddenactivities', $context)) {
        notice(get_string("activityiscurrentlyhidden"));
    }

    //check for groupmode
    $groupmode = groupmode($course, $cm);
    $currentgroup = get_and_set_current_group($course, $groupmode,
                                              $changegroup);

    if ($groupmode == SEPARATEGROUPS && ($currentgroup === false) &&
            !has_capability('moodle/site:accessallgroups', $context)) {
        notice(get_string('notingroup', 'mumiemodule'));
    }
    /// Okay, we can show the grades. Log the mumiemodule view.
    if ($cm->id) {
        add_to_log($course->id, "mumiemodule", "view grades",
                   "view.php?id=$cm->id", "$mumiemodule->id", $cm->id);
    } else {
        add_to_log($course->id, "mumiemodule", "view grades",
                    "view.php?f=$forum->id", "$mumiemodule->id");
    }
```

```
    /// Now we need a menu for separategroups as well!
    if ($groupmode == VISIBLEGROUPS || ($groupmode
            && has_capability('moodle/site:accessallgroups', $context))) {

       if($groups = get_groups($course->id)){
            print_box_start('groupmenu');
            print_group_menu($groups, $groupmode, $currentgroup,
                          "$CFG->wwwroot/mod/mumiemodule/view.php?id=$cm->id");
            print_box_end(); // groupmenu
        }
    }/// Only print menus the student is in any course
    else if ($groupmode == SEPARATEGROUPS){
        $validgroups = array();
        // Get all the groups this guy is in in this course

        if ($p = user_group($course->id,$USER->id)){
            /// Extract the name and id for the group
            foreach ($p as $index => $object){
                $validgroups[$object->id] = $object->name;
            }
            /// Print them in the menu
            print_box_start('groupmenu');
            print_group_menu($validgroups, $groupmode, $currentgroup,
                             "view.php?id=$cm->id",0);
            print_box_end(); // groupmenu
        }
    }
            echo '<br />';
            if (!empty($showall)) {
                show_grades($currentgroup, $cm, $course, $mumiemodule);
            } else {
                show_grades($currentgroup, $cm, $course, $mumiemodule);
            }
}else if(has_capability('mod/mumiemodule:participate', $context)){

    if (!record_exists('mumiemodule_students', 'mumiemodule', $mumiemodule->id,
                       'userid', $USER->id)){
        error ("You have to take part in this activity to take a look!
               So you should be part of a group.",
               "../../course/view.php?id=".$COURSE->id);
    }
    $groupmode = groupmode($course, $cm);
    $currentgroup = get_and_set_current_group($course, $groupmode,
                                              $changegroup);

    if ($groupmode == SEPARATEGROUPS && ($currentgroup === false) &&
    !has_capability('moodle/site:accessallgroups', $context)) {
        notify(get_string('notingroupstudent', 'mumiemodule'));
    }

    $currentgrade = get_field('mumiemodule_students', 'grade', 'mumiemodule',
                              $mumiemodule->id, 'userid', $USER->id);
    $currentgrade .= ' / '.$mumiemodule->grade;
    print_box("Current points: ".$currentgrade);
}else {
   error("You do not have the right permission!");
}
```

```
/// Finish the page
    print_footer($course);

    /**
     * Function for showing the students points to a creator, teacher or trainer
     */
     function show_grades($currentgroup, $cm, $course, $mumiemodule){
        global $CFG, $USER, $db;

        //needed? I am not sure
        $page    = optional_param('page', 0, PARAM_INT);

        /// Get all teachers and students
        if ($currentgroup) {
            $users = get_records('mumiemodule_students', 'groupid',
                                 $currentgroup, '', 'userid');
            //$users = get_group_users($currentgroup);
        } else {
            $context = get_context_instance(CONTEXT_MODULE, $cm->id);
            $users = get_users_by_capability($context,
                                             'mod/mumiemodule:participate');
                        // everyone with this capability set to non-prohibit
            //TODO we need a user-list with just students that are in groups
        }

        $tablecolumns = array('picture', 'fullname', 'grade', 'timemodified');
        $tableheaders = array('', get_string('fullname'),
                              get_string('grade', 'mumiemodule'),
                              get_string('lastmodified', 'mumiemodule')
                                                  .' ('.$course->student.')');

        require_once($CFG->libdir.'/tablelib.php');
        $table = new flexible_table('mumiemodule-current-grades');

        $table->define_columns($tablecolumns);
        $table->define_headers($tableheaders);
        $table->define_baseurl($CFG->wwwroot
                                    .'/mod/mumiemodule/view.php?id='
                                    .$cm->id
                                    .'&amp;currentgroup='.$currentgroup);

        $table->sortable(true, 'lastname');//sorted by lastname by default
        $table->collapsible(true);
        $table->initialbars(true);

        $table->column_suppress('picture');
        $table->column_suppress('fullname');

        $table->column_class('picture', 'picture');
        $table->column_class('fullname', 'fullname');
        $table->column_class('grade', 'grade');
        $table->column_class('timemodified', 'timemodified');

        $table->set_attribute('cellspacing', '0');
        $table->set_attribute('id', 'attempts');
        $table->set_attribute('class', 'grades');
        $table->set_attribute('width', '90%');
        $table->set_attribute('align', 'center');
```

```
    // Start working -- this is necessary as soon as the niceties are over
    $table->setup();

/// Check to see if groups are being used in this assignment

    /*if (!$teacherattempts) { //@toDo: check variable!!!
        $teachers = get_course_teachers($course->id);
        if (!empty($teachers)) {
            $keys = array_keys($teachers);
        }
        foreach ($keys as $key) {
            unset($users[$key]);
        }
    }*/

     /// Construct the SQL

    if ($where = $table->get_sql_where()) {
        $where .= ' AND ';
    }

    if ($sort = $table->get_sql_sort()) {
        $sort = ' ORDER BY '.$sort;
    }

    $select = 'SELECT u.id, u.firstname, u.lastname, u.picture,
                      s.id AS mumiepartsid, s.grade,
                      s.timemodified ';
    $sql = 'FROM '.$CFG->prefix.'user u '.
           'LEFT JOIN '
             .$CFG->prefix.'mumiemodule_students s
             ON u.id = s.userid
             AND s.mumiemodule = '.$mumiemodule->id.' '.
           'WHERE '.$where.'u.id IN ('.implode(',', array_keys($users)).')';

    $table->pagesize(10, count($users));

    //offset used to calculate index of student in that particular query
    //needed for the pop up to know who's next
    $offset = $page * 10;
    $strupdate = get_string('update');
    $strgrade  = get_string('grade');
    $grademenu = make_grades_menu($mumiemodule->grade);

    if (($ausers = get_records_sql($select.$sql.$sort,
          $table->get_page_start(), $table->get_page_size())) !== false) {
    //if (($ausers = get_records($select.$sql.$sort,
            $table->get_page_start(), $table->get_page_size())) !== false) {

        foreach ($ausers as $auser) {
        // Calculate user status
            $auser->status = 1;
            $picture = print_user_picture($auser->id, $course->id,
                                          $auser->picture, false, true);

                $studentmodified = '<div id="ts'.$auser->id.'">'
                       ./*$this->print_student_answer($auser->id)
                       .*/userdate($auser->timemodified).'</div>';
```

```
                $grade = '<div id="g'.$auser->id.'">'
                           .$auser->grade.'/'.$mumiemodule->grade.'</div>';

            $buttontext = ($auser->status == 1) ? $strupdate : $strgrade;
            //No more buttons, we use popups ;-).
            $button = link_to_popup_window('/mod/mumiemodule/view.php?
                                           id='.$cm->id
                                           .'&amp;userid='
                                           .$auser->id
                                           .'&amp;mode=single'
                                           .'&amp;offset='.$offset++,
                                            'grade'.$auser->id, $buttontext,
                                           500, 780, $buttontext, 'none',
                                           true, 'button'.$auser->id);

            $status  = '<div id="up'.$auser->id.'" class="s'
                           .$auser->status.'">'.$button.'</div>';

            $row = array($picture, fullname($auser), $grade,
                         $studentmodified);
            $table->add_data($row);
        }
    }

 /**
 *  Return a grade in user-friendly form, whether it's a scale or not
 *
 * @param $grade
 * @return string User-friendly representation of grade
 */
function display_grade($grade, $mumiemodule) {

    $scalegrades = array();

    if ($mumiemodule->grade >= 0) {    // Normal number
        if ($grade == -1) {
            return '-';
        } else {
            return $grade.' / '.$mumiemodule->grade;
        }

    } else {                                // Scale
        if (empty($scalegrades[$mumiemodule->id])) {
           if ($scale = get_record('scale', 'id', -($mumiemodule->grade))) {
                $scalegrades[$mumiemodule->id] = make_menu_from_list
                                                    ($scale->scale);
            } else {
                return '-';
            }
        }
        if (isset($scalegrades[$mumiemodule->id][$grade])) {
            return $scalegrades[$mumiemodule->id][$grade];
        }
        return '-';
    }
}
 $table->print_html();  /// Print the whole table
 }
?>
```

A1.4 Quellcode config.html

```
<form method="post" action="module.php" id="form">
<input type="hidden" name="sesskey" value="<?php echo $USER->sesskey ?>">

<table cellpadding="9" cellspacing="0">
<tr valign="top">
    <td align="right">syncServer:</td>
    <td> <input name="syncServer" type="text" size="50" value="<?php if
(isset($CFG->syncServer)) p($CFG->syncServer) ?>" /></td>
    <td>
    <?php print_string('syncServer', 'mumiemodule') ?>
    </td>
</tr>
<!--
<tr valign="top">
    <td align="right">syncServer:</td>
    <td> <input name="syncServer" type="text" size="50" value="<?php if
(isset($CFG->syncServer)) p($CFG->syncServer) ?>" /></td>
    <td>
    <?php /*print_string('syncServer', 'mumiemodule')*/ ?>
    </td>
</tr>
-->
<tr valign="top">
    <td align="right">syncUser:</td>
    <td> <input name="syncUser" type="text" size="50" value="<?php if
(isset($CFG->syncUser)) p($CFG->syncUser) ?>" />    </td>
    <td>
    <?php print_string('syncUser', 'mumiemodule') ?>
    </td>
</tr>
<tr valign="top">
    <td align="right">syncPassword:</td>
    <td> <input name="syncPassword" type="password" size="50" value="<?php if
(isset($CFG->syncPassword)) p($CFG->syncPassword) ?>" />    </td>
    <td>
    <?php print_string('syncPassword', 'mumiemodule') ?>
    </td>
</tr>

<tr>
    <td colspan="3" align="center">
    <input type="submit" value="<?php print_string("savechanges") ?>"></td>
</tr>
</table>

</form>
```

A1.5 Quellcode lib.php

```
<?php  // $Id: lib.php,v 1.4 2006/08/28 16:41:20 mark-nielsen Exp $
/**
 * Library of functions and constants for module mumiemodule
 *
 * @author PR
 * @version $Id: lib.php
 * @package mumiemodule
 **/


global $CFG;
require_once $CFG->dirroot.'/mod/mumiemodule/japs_sync/hooks.php'; //for the
external synchronisation
require_once $CFG->dirroot.'/mod/mumiemodule/japs_sync/transfer_objects.php';

/**
 * Given an object containing all the necessary data,
 * (defined by the form in mod_form.php) this function
 * will create a new instance and return the id number
 * of the new instance.
 *
 * @param object $mumiemodule - An object from the form in mod.html
 * @return int - The id of the newly inserted newmodule record
 **/
function mumiemodule_add_instance($mumiemodule, $checkfirst=true,
                                  $perhand=false) {

    global $CFG;
    global $COURSE;
    global $USER;

    //check if this is not the firs mumiemodule within this course:
    $morethanone = record_exists('mumiemodule', 'course', $mumiemodule->course);

    $mumiemodule->timemodified = time();

        //this is just for later-versions:
        if(!$morethanone){
            $thebigone = create_the_big_one($mumiemodule);
            //to create the big object containing all data
            insert_all_for_mumie($thebigone);
            } else {
                //so there are more than one MUMIE-courses related to that class
                //we dont want to have more than one mumiemodule - perhaps later
                //TODO:
                //del errormessage when adding another mumiemodule-instance
                //rebuild sending of mumie-courses
            }

        //insert the new activity into Moodle-DB:
        $insertedid = insert_record("mumiemodule", $mumiemodule);
        //and the students
        $context = get_context_instance(CONTEXT_COURSE, $COURSE->id);
        $users = get_users_by_capability($context,
                                         'mod/mumiemodule:participate');
```

```
        if ($mumie_groups = get_groups($COURSE->id)){
            foreach($mumie_groups as $mumie_group){
                foreach ($users as $user) {
                    if(groups_is_member($mumie_group->id, $user->id)){
                        $ins = new object();
                        $ins->mumiemodule = $insertedid;
                        $ins->userid  = $user->id;
                        $ins->groupid = (int)$mumie_group->id;
                        $ins->timemodified = $mumiemodule->timemodified;
                        $newid = insert_record("mumiemodule_students", $ins);
                    }
                }
            }
        }
    if(!$perhand){
        return $insertedid;
    }else{
        echo '<h1 align="center">Done!</h1>';
        print_continue($CFG->wwwroot);
        admin_externalpage_print_footer();
        exit;
    }
}

/**
 * Given an object containing all the necessary data,
 * (defined by the form in mod_form.php) this function
 * will update an existing instance with new data.
 *
 * @param object $mumiemodule - An object from the form in mod_form.php
 * @return boolean Success/Fail
 **/
function mumiemodule_update_instance($mumiemodule) {

    $mumiemodule->timemodified = time();
    $mumiemodule->id = $mumiemodule->instance;

    return update_record("mumiemodule", $mumiemodule);
    //no need to inform MUMIE - just Moodle-internal changes possible here
}

/**
 * Given an ID of an instance of this module,
 * this function will permanently delete the instance
 * and any data that depends on it.
 *
 * @param int $id - Id of the module instance
 * @return boolean Success/Failure
 **/
function mumiemodule_delete_instance($id) {

    if (! $mumiemodule = get_record("mumiemodule", "id", "$id")) {
        return false;
    }

    $result = true;

    if (! delete_records("mumiemodule", "id", "$mumiemodule->id")) {
        $result = false;
```

```
    } else { //delete elements from referenced table
        if (!delete_records("mumiemodule_students", "mumiemodule",
                            $mumiemodule->id)){
            $result = false;
        }
    }

   if($result){
    //TODO:
    //send to MUMIE when sync-order is implemented there
    }

    return $result;
}

/**
 * Return a small object with summary information about what a
 * user has done with a given particular instance of this module
 * Used for user activity reports.
 * $return->time = the time they did it
 * $return->info = a short text description
 *
 * @return null
 * @todo Finish documenting this function
 **/
function mumiemodule_user_outline($course, $user, $mod, $mumiemodule) {
    //TODO:
    return null;

}

/**
 * Print a detailed representation of what a user has done with
 * a given particular instance of this module, for user activity reports.
 *
 * @return boolean
 * @todo Finish documenting this function
 **/
function mumiemodule_user_complete($course, $user, $mod, $mumiemodule) {
    //TODO:
    return true;
}

/**
 * Given a course and a time, this module should find recent activity
 * that has occurred in mumiemodule activities and print it out.
 * Return true if there was output, or false is there was none.
 *
 * @uses $CFG
 * @return boolean
 * @todo Finish documenting this function
 **/
function mumiemodule_print_recent_activity($course, $isteacher, $timestart) {
    global $CFG;
    //TODO:
    return false;  //  True if anything was printed, otherwise false
}
```

```
/**
 * Function to be run periodically according to the moodle cron
 * This function searches for things that need to be done, such
 * as sending out mail, toggling flags etc ...
 *
 * @uses $CFG
 * @return boolean
 * @todo Finish documenting this function
 **/
function mumiemodule_cron () {
    global $CFG;
    return true;
}

/**
 * Must return an array of grades for a given instance of this module,
 * indexed by user.  It also returns a maximum allowed grade.
 *
 * Example:
 *    $return->grades = array of grades;
 *    $return->maxgrade = maximum allowed grade;
 *
 *    return $return;
 *
 * @param int $mumiemoduleID of an instance of this module
 * @return mixed Null or object with an array of grades and with the maximum
 *   grade
 **/
function mumiemodule_grades($mumiemoduleid) {
   if (!$mumiemodule = get_record('mumiemodule', 'id', $mumiemoduleid)) {
        return NULL;
    }
    if ($mumiemodule->grade == 0) { // No grading
        return NULL;
    }

    $grades = get_records_menu('mumiemodule_students', 'mumiemodule',
                               $mumiemodule>id, '', 'userid, grade');

    $return = new object();

    if ($mumiemodule->grade > 0) {
        if ($grades) {
            foreach ($grades as $userid => $grade) {
                if ($grade == -1) {
                    $grades[$userid] = '-';
                }
            }
        }
        $return->grades = $grades;
        $return->maxgrade = (int)$mumiemodule->grade;

    } else { // Scale
        if ($grades) {
            $scaleid = - ($mumiemodule->grade);
            $maxgrade = "";
            if ($scale = get_record('scale', 'id', $scaleid)) {
                $scalegrades = make_menu_from_list($scale->scale);
                foreach ($grades as $userid => $grade) {
```

```
                    if (empty($scalegrades[$grade])) {
                        $grades[$userid] = '-';
                    } else {
                        $grades[$userid] = $scalegrades[$grade];
                    }
                }
                $maxgrade = $scale->name;
            }
        }
        $return->grades = $grades;
        $return->maxgrade = $maxgrade;
    }

    return $return;
}

/**
 * Must return an array of user records (all data) who are participants
 * for a given instance of mumiemodule. Must include every user involved
 * in the instance, independient of his role (student, teacher, admin...)
 * See other modules as example.
 *
 * @param int $mumiemoduleid ID of an instance of this module
 * @return mixed boolean/array of students
 **/
function mumiemodule_get_participants($mumiemoduleid) {
    global $CFG;

/*
    //Get students
    $students = get_records_sql("SELECT DISTINCT u.id, u.id
                                 FROM {$CFG->prefix}user u,
                                      {$CFG->prefix}mumiemodule_students a
                                 WHERE a.mumiemodule = '$mumiemoduleid' and
                                       u.id = a.userid");
    //Get teachers
    $teachers = get_records_sql("SELECT DISTINCT u.id, u.id
                                 FROM {$CFG->prefix}user u,
                                      {$CFG->prefix}mumiemodule_students a
                                 WHERE a.mumiemodule = '$mumiemoduleid' and
                                       u.id = a.teacher");

    //Add teachers to students
    if ($teachers) {
        foreach ($teachers as $teacher) {
            $students[$teacher->id] = $teacher;
        }
    }
    //Return students array (it contains an array of unique users)
    return ($students);
    */
    return false;
}
```

```
/**
 * This function returns if a scale is being used by one module
 * it it has support for grading and scales. Commented code should be
 * modified if necessary. See forum, glossary or journal modules
 * as reference.
 *
 * @param int $newmoduleid ID of an instance of this module
 * @return mixed
 * @todo Finish documenting this function
 **/
function mumiemodule_scale_used ($mumiemoduleid,$scaleid) {
    $return = false;

    //$rec = get_record("newmodule","id","$newmoduleid","scale","-$scaleid");
    //
    //if (!empty($rec)  && !empty($scaleid)) {
    //    $return = true;
    //}

    return $return;
}

 /**
 * Adds students to the mumiemodule
 *
 * @param integer $userid - the ID of the user
 * @param integer $mumiemodule - the ID of the mumiemodule
 * @param bool $morethanone - true if there is at least one more mumiemodule-
 *         instance within this course
 * @param object $mumie_group - the group in which to subscribe
 */
function mumiemodule_students_subscribe($userid, $mumiemoduleid, $mumie_group,
                                        $morethanone = false) {
    global $CFG;
    global $COURSE;

    if (record_exists("mumiemodule_students", "mumiemodule", $mumiemoduleid,
            ^           "userid", $userid, "group", $mumie_group->id)) {
        return true;
    }

    $ins = new object();
    $ins->mumiemodule = $mumiemoduleid;
    $ins->userid  = $userid;
    $ins->groupid = $mumie_group->id;
    $ins->timemodified = time();

     //does the MUMIE already know the user?
    $studenttocheck = new object();
    $studenttocheck->syncid = 'moodle-'.$CFG->prefix.'user-'.$userid;
    $userexists = check_student($studenttocheck);

    //now we can insert the user into DB
    $newid = insert_record("mumiemodule_students", $ins);

    if($newid){
        if(!$morethanone && !$userexists){ //MUMIE does not know this user yet
            $newuser = get_record('user', 'id', $userid);
            send_single_user_to_mumie($newuser);
```

```
        }
        $group_sync_id = 'moodle-'.$CFG->prefix.'groups-'.$mumie_group->id;
        $user_sync_id = 'moodle-'.$CFG->prefix.'user-'.$newuser->id;
        add_user_to_mumie_tutorial($user_sync_id, $group_sync_id);
    }
    return $newid;
}

/**
 * Remove students from a mumiemodule
 *
 * @param integer $userid
 * @param integer $mumiemoduleid
 * @param integer $groupid
 */
 function mumiemodule_students_unsubscribe($userid, $mumiemoduleid, $groupid){

    if (!record_exists("mumiemodule_students", "mumiemodule", $mumiemoduleid,
                        "userid", $userid, "groupid", $groupid)) {
        return true;
    }

    return delete_records("mumiemodule_students", "userid", $userid,
                          "mumiemodule", $mumiemoduleid, "groupid", $groupid);
 }

/**
 * This function gets run whenever a role is assigned to a user in a context
 *
 * @param integer $userid
 * @param object $context
 * @return bool
 */
function mumiemodule_role_assign($userid, $context) {
    //here we just want to insert lecturers
    if(has_capability('mod/mumiemodule:teachcourse', $context, $userid)){
        return mumiemodule_add_teacher_subscriptions($userid, $context);
    }
    return true;
}


/**
 * This function gets run whenever a role is assigned to a user in a context
 *
 * @param integer $userid
 * @param object $context
 * @return bool
 */
function mumiemodule_role_unassign($userid, $context) {
    global $CFG;
    //it is not possible to check if this had been a teacher before removal
    //but we can check if this user was NOT a student in a group in this course
    if($context->contextlevel==CONTEXT_COURSE){
        if($groups = get_groups($context->instanceid, $userid)){
            foreach($groups as $group){
                if(groups_is_member($group->id, $userid)){
                    //so this is a tutor or student
```

```
                    $mumiemodules = get_records('mumiemodule', 'course',
                                                $context->instanceid);
                                    //list of all mumiemodules in the course
                    $mumiemodules = array_values($mumiemodules);
                        if(!record_exists('mumiemodule_students', 'userid',
                                          $userid, 'mumiemodule',
                                          $mumiemodules[0]->id)){
                                          //so this was no student
                            //the user had been a tutor
                            //we have to change the tutorial
                            $tutorial = new object();
                            $tutorial->syncid = 'moodle-'
                                              .$CFG->prefix.'group-'
                                              .$group->id;
                            $tutorial->tutor = 'moodle-dummy_tutor';
                            change_tutorial_for_mumie($tutorial);
                           //the user could have been a tutor
                           //but could also been a teacher - so try:
                           return mumiemodule_remove_teacher_subscriptions
                                    ($userid, $context);
                        } else { //this had been a student
                                 //that is still in a MUMIE-tutorial
                            $user_sync_id = 'moodle-'
                                              .$CFG->prefix.'user-'.$userid;
                            $group_sync_id = 'moodle-'
                                              .$CFG->prefix.'group-'
                                              .$group->id;
                            remove_user_from_mumie_tutorial($user_sync_id,
                                                            $group_sync_id);
                            //and now the student has to be removed
                            //from all mumiemodules in this course
                            foreach ($mumiemodules as $mumiemodule){
                                mumiemodule_students_unsubscribe($userid,
                                                                 $mumiemodule,
                                                                 $group->id);
                            }
                        }
                }
            }
        } else { //this user could have been nearly any role - so try:
            return mumiemodule_remove_teacher_subscriptions($userid, $context);
        }
    }
    //for any other context we try to remove teachers:
    return mumiemodule_remove_teacher_subscriptions($userid, $context);
}
```

```
/**
 * Recursive function to add subscriptions for new teachers
 * students are just assigned via groups
 *
 * @param $userid - the ID of the user added
 * @param $context - the context the user is added in
 * @param roleid - the ID of the user's role
 */
function mumiemodule_add_teacher_subscriptions($userid, $context) {
    global $CFG;

    if (empty($context->contextlevel)) {
        return false;
    }

    switch ($context->contextlevel) {

        case CONTEXT_SYSTEM:   // For the whole site
             if ($courses = get_records('course')) {
                 foreach ($courses as $course) {
                     $subcontext = get_context_instance(CONTEXT_COURSE,
                                                        $course->id);
                     mumiemodule_add_teacher_subscriptions($userid,
                                                           $subcontext);
                 }
             }
             break;

        case CONTEXT_COURSECAT:   // For a whole category
             if ($courses = get_records('course', 'category',
                                        $context->instanceid)) {
                 foreach ($courses as $course) {
                     $subcontext = get_context_instance(CONTEXT_COURSE,
                                                        $course->id);
                     mumiemodule_add_teacher_subscriptions($userid,
                                                           $subcontext);
                 }
             }
             if ($categories = get_records('course_categories', 'parent',
                                           $context->instanceid)) {
                 foreach ($categories as $category) {
                     $subcontext = get_context_instance(CONTEXT_COURSECAT,
                                                        $category->id);
                     mumiemodule_add_teacher_subscriptions($userid,
                                                           $subcontext);
                 }
             }
             break;


        case CONTEXT_COURSE:   // For a whole course
             if ($course = get_record('course', 'id', $context->instanceid)) {
                 if ($mumiemodules = get_all_instances_in_course('mumiemodule',
                                                                 $course,
                                                                 $userid,
                                                                 false)) {
                     $fields = 'id, username, password, firstname, lastname';
                     //check if this lecturer is already in mumie as user
                     $lecturertocheck = new object();
```

```
                $lecturertocheck->syncid = 'moodle-'
                                          .$CFG->prefix.'user-'.$userid;
                $lecturerexists = check_lecturer($lecturertocheck,
                                                 $course->id);
                if(!$lecturerexists){
                   $newuser = get_record('user', 'id', $userid, '', '', '',
                                         '', $fields);
                   send_single_user_to_mumie($newuser);
                }
                foreach ($mumiemodules as $mumiemodule) {
                    if ($modcontext = get_context_instance(CONTEXT_MODULE,
                         $mumiemodule->coursemodule)) {
                       $course_to_change = new object();
                       $course_to_change->syncid = 'moodle-'
                                                   .$CFG->prefix
                                                   .'course-'.$course->id;
                       $course_to_change->name = $course->fullname;
                       $course_to_change->description = $course->summary;
                       $course_to_change->semester = 'moodle-'
                                                     .$CFG->prefix
                                                     .'course_categories-'
                                                     .$course->category;
                       $course_to_change->lecturers = '';
                       //$context = get_context_instance(CONTEXT_COURSE,
                                                         $course->id);
                       $roleid = get_field('role', 'id', 'shortname',
                                           'editingteacher');
                       $fields = 'u.id, u.username, u.password,
                                  u.firstname, u.lastname';
                       if ($lecturers = get_role_users($roleid, $context,
                                                       false, $fields)){
                           foreach($lecturers as $lecturer){
                               //get mumie to know the lecturer:
                               //send_single_user_to_mumie($lecturer);
                               //MUMIE should already know them all
                               $course_to_change->lecturers .=
                                                     'moodle-'
                                                     .$CFG->prefix
                                                     .'user-'
                                                     .$lecturer->id.' ';
                           }
                       }
                       //$course_to_change->lecturers .= 'moodle-'
                                                     .$CFG->prefix
                                                     .'user-'.$userid;
                                                 //or he is in the list twice
                       change_class_for_mumie($course_to_change);
                    }
                }
            }
        }
        break;

//is this relevant?? can we really assign a role to just one mumiemodule
case CONTEXT_MODULE:   // Just one module
                      //necessary for mumiemodule???
```

```
            if ($cm = get_coursemodule_from_id('mumiemodule',
                                               $context->instanceid)) {
                if ($mumiemodule = get_record('mumiemodule', 'id',
                                               $cm->instance)) {
                    if (has_capability('mod/mumiemodule:tutorize',
                                       $modcontext, $userid)){
                                $newuser = get_record('user', 'id', $userid);
                                send_single_user_to_mumie($newuser);
                    }
                }
            }
            break;
    }


    return true;
}


/**
 * Recursive function to remove subscriptions for a teacher in a context
 */
function mumiemodule_remove_teacher_subscriptions($userid, $context) {
    global $CFG;

    if (empty($context->contextlevel)) {
        return false;
    }

    switch ($context->contextlevel) {

        case CONTEXT_SYSTEM:   // For the whole site
            if ($courses = get_records('course')) {
                foreach ($courses as $course) {
                    $subcontext = get_context_instance(CONTEXT_COURSE,
                                                       $course->id);
                    mumiemodule_remove_user_subscriptions($userid, $subcontext);
                }
            }
            break;

        case CONTEXT_COURSECAT:   // For a whole category
             if ($courses = get_records('course', 'category',
                                        $context->instanceid)) {
                foreach ($courses as $course) {
                    $subcontext = get_context_instance(CONTEXT_COURSE,
                                                       $course->id);
                    mumiemodule_remove_user_subscriptions($userid,
                                                       $subcontext);
                }
            }
            if ($categories = get_records('course_categories', 'parent',
                                          $context->instanceid)) {
                foreach ($categories as $category) {
                    $subcontext = get_context_instance(CONTEXT_COURSECAT,
                                                       $category->id);
                    mumiemodule_remove_user_subscriptions($userid,
                                                       $subcontext);
                }
```

```
            }
            break;

        case CONTEXT_COURSE:   // For a whole course
            if ($course = get_record('course', 'id', $context->instanceid)) {
                if ($mumiemodules = get_all_instances_in_course('mumiemodule',
                                                    $course, $userid, true)) {
                    foreach ($mumiemodules as $mumiemodule) {
                        if ($modcontext = get_context_instance(CONTEXT_MODULE,
                                                $mumiemodule->coursemodule)) {
                            $course_to_change = new object();
                            $course_to_change->syncid = 'moodle-'
                                                    .$CFG->prefix
                                                    .'course-'.$course->id;
                            $course_to_change->name = $course->fullname;
                            $course_to_change->description = $course->summary;
                            $course_to_change->semester = 'moodle-'
                                                    .$CFG->prefix
                                                    .'course_categories-'
                                                    .$course->category;
                            $course_to_change->lecturers = '';
                            $context = get_context_instance(CONTEXT_COURSE,
                                                    $course->id);
                            $fields = 'u.id, u.username, u.password,
                                        u.firstname, u.lastname';
                            $roleid = get_field('role', 'id', 'shortname',
                                                'editingteacher');
                            if ($lecturers = get_role_users($roleid, $context,
                                                    false, $fields)){
                               foreach($lecturers as $lecturer){
                                   if($userid!=$lecturer->id){
                                       $course_to_change->lecturers .=
                                                    'moodle-'
                                                    .$CFG->prefix
                                                    .'user-'
                                                    .$lecturer->id
                                                    .' ';
                                   }
                               }
                            }
                            change_class_for_mumie($course_to_change);
                        }
                    }
                }
            }
            break;

        case CONTEXT_MODULE:   // Just one mumiemodule
            if ($cm = get_coursemodule_from_id('mumiemodule',
                                        $context->instanceid)) {
                if ($mumiemodule = get_record('mumiemodule', 'id',
                                        $cm->instance)) {
                    //nix zu tun, da keine teacher im modul stehen
                }
            }
            break;
    }
    return true;
}
```

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//                  Functions to send the information to HOOKS
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

    /*function send_new_semester_to_mumie(){
        global $CFG;
        global $COURSE;
        global $USER;
        //zuerst das Semester:
        $semester = new object();
        $semester->syncid = 'moodle-'.$CFG->prefix.'course_categories-'
                            .$COURSE->category;
        $semester->name = get_field('course_categories', 'name', 'id',
                                    $COURSE->category);
        $semester->description = 'a Moodle-category as semester';
        insert_semester_for_mumie($semester);
    }*/

    /*function send_new_class_to_mumie(){
        global $CFG;
        global $COURSE;
        global $USER;
        //nun die LV:
        $class = new object();
        $class->syncid = 'moodle-'.$CFG->prefix.'course-'.$COURSE->id;
        $class->name = $COURSE->fullname;
        $class->description = $COURSE->summary;
        $class->semester = 'moodle-'.$CFG->prefix.'course_categories-'
                           .$COURSE->category;
        $class->lecturers = '';
        $context = get_context_instance(CONTEXT_COURSE, $COURSE->id);
        $fields = 'u.id, u.username, u.password, u.firstname, u.lastname';
        $roleid = get_field('role', 'id', 'shortname', 'editingteacher');
        if ($lecturers = get_role_users($roleid, $context, false, $fields)){
            foreach($lecturers as $lecturer){
                //get mumie to know the lecturer:
                send_single_user_to_mumie($lecturer);
                $class->lecturers .= 'moodle-'.$CFG->prefix.'user-'
                                     .$lecturer->id.' ';
            }
        }
        insert_class_for_mumie($class);
    }*/

    function send_new_tutorial_to_mumie($mumie_group){
                global $CFG;
                global $COURSE;
                global $USER;
                $tutorial = new object();
                $tutorial->syncid = 'moodle-'
                                    .$CFG->prefix.'groups-'.$mumie_group->id;
                $tutorial->name = $mumie_group->name;
                $tutorial->description = $mumie_group->description;
                $tutorial->tutor = 'moodle-dummy_tutor';
```

```
            $tutorial->classid = 'moodle-'
                                 .$CFG->prefix.'course-'
                                 .$mumie_group->courseid;
            insert_tutorial_for_mumie($tutorial);
}

/*
function send_students_to_mumie(){
    //jetzt die studenten:
    $context = get_context_instance(CONTEXT_COURSE, $COURSE->id);
    $fields = 'u.id, u.username, u.password, u.firstname, u.lastname';
    $mumie_users = get_users_by_capability($context,
                                           'mod/mumiemodule:participate',
                                           $fields);
    if(!empty($mumie_users)){
        foreach($mumie_users as $mumie_user){
            send_single_user_to_mumie($mumie_user);

            if ($mumie_groups = get_groups($COURSE->id)){
                //user zu tutorien hinzufuegen
                foreach ($mumie_groups as $mumie_group){
                    //if(ismember($mummie_group->id, $mumie_user->id)){
                    if (groups_is_member($mumie_group->id,
                                         $mumie_user->id)){
                        $group_sync_id = 'moodle-'
                                         .$CFG->prefix
                                         .'course_categories-'
                                         .$mumie_group->id;
                        $user_sync_id = 'moodle-'
                                        .$CFG->prefix.'user-'.$mumie_user->id;
                        add_user_to_mumie_tutorial($user_sync_id,
                                                   $group_sync_id);
                    }
                }
            }
        }
    }
}
*/

function send_single_user_to_mumie($mumie_user){
            global $CFG;
            global $COURSE;
            global $USER;
            //user:
            $newuser = new object();
            $newuser->syncid = 'moodle-'.$CFG->prefix.'user-'
                                        .$mumie_user->id;
            $newuser->loginname = $mumie_user->username;
            $newuser->passwordencrypted = $mumie_user->password;
            $newuser->firstname = $mumie_user->firstname;
            $newuser->surname = $mumie_user->lastname;
            $newuser->matrnumber = 'X';
            insert_user_for_mumie($newuser);
}
```

```
    ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
    ////      New function to create the XML-objects for the interface
/////////////////
    ////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////

    function get_mumie_user($user){
                global $CFG;
                $newuser = new object();
                $newuser->syncid = 'moodle-'.$CFG->prefix.'user-'.$user->id;
                $newuser->loginname = $user->username;
                $newuser->passwordencrypted = $user->password;
                $newuser->firstname = $user->firstname;
                $newuser->surname = $user->lastname;
                $newuser->matrnumber = 'X';
                return $newuser;
    }

    function create_the_big_one($mumiemodule){
        global $CFG;
        global $USER;
        global $COURSE;

        //first the semester
        $semester_id = 'moodle-'.$CFG->prefix.'course_categories-'
                        .$COURSE->category;
        $semester_name = get_field('course_categories', 'name', 'id',
                                   $COURSE->category);
                          //TODO: replace this hack by api-method when available
        $semester_description = 'a Moodle-category as semester';
        $semester = new Semester($semester_id, $semester_name,
                                 $semester_description);

        //lets get the lecturers
        $context = get_context_instance(CONTEXT_COURSE, $COURSE->id);
        $fields = 'u.id, u.username, u.password, u.firstname, u.lastname';
        $lecturers = null;
        $roleid = get_field('role', 'id', 'shortname', 'editingteacher');
        if ($mumie_users = get_role_users($roleid, $context, true, $fields)){
            $lecturers = array();
            foreach($mumie_users as $mumie_user){
                $newuser = get_mumie_user($mumie_user);
                $lecturer = new Lecturer($newuser->syncid, $newuser->loginname,
                                         $newuser->passwordencrypted,
                                         $newuser->firstname, $newuser->surname,
                                         $newuser->matrnumber);
                array_push($lecturers, $lecturer);
            }
        }

        //now the class
        $class_id = 'moodle-'.$CFG->prefix.'course-'.$COURSE->id;
        $class_name = $COURSE->fullname;
        $class_description = $COURSE->summary;

        //and now we have the tutorials:
        $tutorials = null;
```

```
if($mumie_groups = get_groups($COURSE->id)){
        $tutorials = array();
        foreach($mumie_groups as $mumie_group){
            $tutor = null;
            $tutorial_id = 'moodle-'.$CFG->prefix
                           .'groups-'.$mumie_group->id;
            $tutorial_name = $mumie_group->name;
            $tutorial_description = $mumie_group->description;
            $context = get_context_instance(CONTEXT_GROUP,
                                            $mumie_group->id);
            $group_users = get_users_by_capability($context,
                                        'mod/mumiemodule:tutorize',
                                        $fields);
            foreach($group_users as $group_user){
                //zur sicherheit, da sonst alle tutoren in jeder gruppe:
                    if(groups_is_member($mumie_group->id,
                                        $group_user->id)){
                            $newuser = get_mumie_user($group_user);
                            $tutor = new Tutor($newuser->syncid,
                                               $newuser->loginname,
                                               $newuser->passwordencrypted,
                                               $newuser->firstname,
                                               $newuser->surname,
                                               $newuser->matrnumber);
                        break;
                      //so just the first tutor will be inserted
                    }
            }
                    //the students within this tutorial:
                    $students = null;
                    if($group_users = get_users_by_capability($context,
                                        'mod/mumiemodule:participate',
                                        $fields)){
                        $students = array();
                        foreach($group_users as $group_user){
                            if(groups_is_member($mumie_group->id,
                                               $group_user->id)){
                                $newuser = get_mumie_user($group_user);
                                $student = new Student($newuser->syncid,
                                                       $newuser->loginname,
                                                  $newuser->passwordencrypted,
                                                   $newuser->firstname,
                                                   $newuser->surname,
                                                   $newuser->matrnumber);
                                array_push($students, $student);
                            }
                        }
                    }
                    //for the transfer_classes:
                    $tutorial = new Tutorial($tutorial_id,
                                             $tutorial_name,
                                             $tutorial_description,
                                             $tutor, $students);
                    array_push($tutorials, $tutorial);
            }
}
```

```
                //don't forget the MUMIE-course
                $mumie_course_id = $mumiemodule->mumie_course_id;
                $mumie_class = new MumieClass($class_id, $class_name,
                                              $class_description, $semester,
                                              $lecturers, $tutorials,
                                              $mumie_course_id);

                return $mumie_class;

    }
?>
```

A1.6 Quellcode install.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<XMLDB PATH="mod/mumiemodule/db" VERSION="20060921" COMMENT="XMLDB file for
Moodle mod/mumiemodule"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../../../lib/xmldb/xmldb.xsd"
      >
  <TABLES>
    <TABLE NAME="mumiemodule" COMMENT="Defines mumiemodules"
      NEXT="mumiemodule_students">
      <FIELDS>
        <FIELD NAME="id" TYPE="int" LENGTH="10" NOTNULL="true" UNSIGNED="true"
            SEQUENCE="true" ENUM="false" NEXT="course"/>
        <FIELD NAME="course" TYPE="int" LENGTH="10" NOTNULL="true"
          UNSIGNED="true" DEFAULT="0" SEQUENCE="false" ENUM="false"
          PREVIOUS="id" NEXT="name"/>
        <FIELD NAME="name" TYPE="char" LENGTH="255" NOTNULL="true"
            SEQUENCE="false" ENUM="false" PREVIOUS="course" NEXT="description"/>
    <FIELD NAME="description" TYPE="text" LENGTH="small" NOTNULL="false"
      SEQUENCE="false" ENUM="false" PREVIOUS="name" NEXT="timemodified"/>
        <FIELD NAME="timemodified" TYPE="int" LENGTH="10" NOTNULL="true"
            UNSIGNED="true" DEFAULT="0" SEQUENCE="false" ENUM="false"
            PREVIOUS="description"/>
      </FIELDS>
      <KEYS>
        <KEY NAME="primary" TYPE="primary" FIELDS="id"
            COMMENT="Primary key for mumiemodule"/>
      </KEYS>
      <INDEXES>
        <INDEX NAME="course" UNIQUE="false" FIELDS="course"/>
      </INDEXES>
    </TABLE>
    <TABLE NAME="mumiemodule_students" COMMENT="Info about current mumiemodules"
      PREVIOUS="mumiemodule">
      <FIELDS>
        <FIELD NAME="id" TYPE="int" LENGTH="10" NOTNULL="true" UNSIGNED="true"
            SEQUENCE="true" ENUM="false" NEXT="mumiemodule"/>
        <FIELD NAME="mumiemodule" TYPE="int" LENGTH="10" NOTNULL="true"
            UNSIGNED="true" DEFAULT="0" SEQUENCE="false" ENUM="false"
            PREVIOUS="id" NEXT="userid"/>
        <FIELD NAME="userid" TYPE="int" LENGTH="10" NOTNULL="true"
            UNSIGNED="true" DEFAULT="0" SEQUENCE="false" ENUM="false"
            PREVIOUS="mumiemodule" NEXT="groupid"/>
        <FIELD NAME="groupid"  TYPE="int" LENGTH="10" NOTNULL="true"
            UNSIGNED="true" DEFAULT="0" SEQUENCE="false" ENUM="false"
            PREVIOUS="userid" NEXT="grade"/>
        <FIELD NAME="grade" TYPE="float" LENGTH="10" NOTNULL="true"
            UNSIGNED="false" DEFAULT="0.00" SEQUENCE="false" ENUM="false"
            DECIMALS="2" PREVIOUS="groupid" NEXT="timemodified"/>
    <FIELD NAME="timemodified" TYPE="int" LENGTH="10" NOTNULL="true"
      UNSIGNED="true" DEFAULT="0" SEQUENCE="false" ENUM="false"
      PREVIOUS="grade"/>
      </FIELDS>
      <KEYS>
        <KEY NAME="primary" TYPE="primary" FIELDS="id" COMMENT="Primary key for
            mumiemodule_students" NEXT="mumiemodule"/>
```

```
        <KEY NAME="mumiemodule" TYPE="foreign" FIELDS="mumiemodule"
            REFTABLE="mumiemodule" REFFIELDS="id" PREVIOUS="primary"/>
      </KEYS>
      <INDEXES>
        <INDEX NAME="userid" UNIQUE="false" FIELDS="userid"/>
      </INDEXES>
    </TABLE>
  </TABLES>

</XMLDB>
```

A1.7 Quellcode access.php

```
<?php // $Id: access.php,v 1.3 2006/10/11 06:22:01 moodler Exp $
/**
 * Capability definitions for the mumiemodule.
 *
 * For naming conventions, see lib/db/access.php.
 */
$mod_mumiemodule_capabilities = array(

    'mod/mumiemodule:teachcourse' => array(

        'captype' => 'write',
        'contextlevel' => CONTEXT_MODULE,
        'legacy' => array(
            'editingteacher' => CAP_ALLOW,
            'coursecreator' => CAP_ALLOW
        )
    ),

    'mod/mumiemodule:participate' => array(

        'captype' => 'write',
        'contextlevel' => CONTEXT_MODULE,
        'legacy' => array(
            'student' => CAP_ALLOW
        )
    ),

    'mod/mumiemodule:tutorize' => array(

        'captype' => 'write',
        'contextlevel' => CONTEXT_MODULE,
        'legacy' => array(
            'editingteacher' => CAP_ALLOW,
            'teacher' => CAP_ALLOW
        )
    ),

    'mod/mumiemodule:manage' => array(

        'captype' => 'write',
        'contextlevel' => CONTEXT_MODULE,
        'legacy' => array(
            'teacher' => CAP_ALLOW,
            'editingteacher' => CAP_ALLOW,
            'coursecreator' => CAP_ALLOW,
            'admin' => CAP_ALLOW
        )
    )
);
?>
```

A1.8 Quellcode events.php

```
<?php // $Id: events.php,v 1.2 2007/06/03 09:10:53 skodak Exp $

$handlers = array (

    'user_updated' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'user_updated_handler',
        'schedule'         => 'instant'
    ),

    'password_changed' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'password_changed_handler',
        'schedule'         => 'instant'
    ),

    'user_deleted' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'user_deleted_handler',
        'schedule'         => 'instant'
    ),

    'course_updated' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'course_updated_handler',
        'schedule'         => 'instant'
    ),

    'course_deleted' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'course_deleted_handler',
        'schedule'         => 'instant'
    ),

    'category_updated' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'category_updated_handler',
        'schedule'         => 'instant'
    ),

    'category_deleted' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'category_deleted_handler',
        'schedule'         => 'instant'
    ),

    'group_created' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'group_created_handler',
        'schedule'         => 'instant'
    ),
```

```
    'group_updated' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'group_updated_handler',
        'schedule'         => 'instant'
    ),

    'group_user_added' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'group_user_added_handler',
        'schedule'         => 'instant'
    ),

    'group_user_removed' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'group_user_removed_handler',
        'schedule'         => 'instant'
    ),

    'group_deleted' => array (
        'handlerfile'      => '/mod/mumiemodule/handler.php',
        'handlerfunction'  => 'group_deleted_handler',
        'schedule'         => 'instant'
    )
);

?>
```

A1.9 Quellcode internalchecks.php

```
<?php
/**
 *
 * script with functions for some internal checks
 * so not everythins has to be send to MUMIE
 */

 /**
  * Function to check if a semester is already known in the MUMIE
  *
  * @param object $semester - the semester to be checked
  */
  function check_semester($semester){
    global $CFG;
    $delimiter = 'moodle-'.$CFG->prefix.'course_categories-';
    $sync_id_parts = explode($delimiter, $semester->syncid);
    $sort = 'c.sortorder ASC';
    $cat_classes = get_courses($sync_id_parts[1], $sort, $fields='c.id');
      //TODO: isn't this deprecated???
    foreach($cat_classes as $cat_class){
        if(record_exists('mumiemodule', 'course', $cat_class->id)){
            return true;
            break;
        }
    }
    return false;
  }

 /**
   * Function to check if a lecturer is already known in the MUMIE
   *
   * @param object $lecturer - the lecturer to be checked
   * @param integer $currentcourseid - the ID of the current course
   * (because there the lecturer is definitely in)
   *
   */
   function check_lecturer($lecturer, $currentcourseid=-1){
        global $CFG;
        $lecturer_delimiter = 'moodle-'.$CFG->prefix.'user-';
        $sync_id_parts = explode($lecturer_delimiter, $lecturer->syncid);
        $classes_in_mumie = get_records('mumiemodule', '', '', '', 'course');
        foreach($classes_in_mumie as $class_in_mumie){
            $coursecontext = get_context_instance(CONTEXT_COURSE,
                                                  $class_in_mumie->course);
            ////////////////////////// Followin is just for the server-test /////
            $sitecontext = get_context_instance(CONTEXT_SYSTEM, SITEID);
            $admin = false;
            if(has_capability('moodle/course:create', $sitecontext,
                              $sync_id_parts[1])){
                $admin = true;
            }
////////////////////////////////////////////////////////////////////////////////
```

```
            if(has_capability('mod/mumiemodule:teachcourse', $coursecontext,
                              $sync_id_parts[1])
                              && $class_in_mumie->course!=$currentcourseid
                              && !$admin){
                return true;
            }
        }
        return false;
    }

    /**
     * Function to check if a tutor is already known in MUMIE
     *
     * @param object $tutor - the tutor to be checked
     *
     */
     function check_tutor($tutor){
         global $CFG;
         $delimiter = 'moodle-'.$CFG->prefix.'user-';
         $sync_id_parts = explode($delimiter, $tutor->syncid);
         $fields = 'u.id';
         $classes_in_mumie = get_records('mumiemodule', '', '', '', 'course');
         foreach($classes_in_mumie as $class_in_mumie){
             $groups_in_class = groups_get_groups($class_in_mumie->course);
             foreach($groups_in_class as $group_in_class){
                 $groupcontext = get_context_instance(CONTEXT_GROUP,
                                                      $group_in_class);
                 ///Following is just for server-test//////////////////////////
                 $sitecontext = get_context_instance(CONTEXT_SYSTEM, SITEID);
                 $admin = has_capability('moodle/course:create', $sitecontext,
                                         $sync_id_parts[1]);
                 ////////////////////////////////////////////////////////////////
                 if(has_capability('mod/mumiemodule:tutorize', $groupcontext,
                                   $sync_id_parts[1]) && !$admin){
                     return true;
                     break;
                 }
             }
         }
         return false;
     }

     /**
      * Function to check if a student is already known in MUMIE
      *
      * @param object $student - the student to be checked
      */
      function check_student($student){
         global $CFG;
         $delimiter = 'moodle-'.$CFG->prefix.'user-';
         $sync_id_parts = explode($delimiter, $student->syncid);
         if(record_exists('mumiemodule_students', 'userid', $sync_id_parts[1])){
             return true;
         }
         return false;
      }


?>
```

A1.10 Quellcode handler.php

```
<?php
/**
 * Library of handler-functions for module mumiemodule
 *
 * @author PR
 * @version $Id: lib.php,v 1.4 2007/06/22
 * @package mumiemodule
 **/

 global $CFG;
 require_once $CFG->dirroot.'/mod/mumiemodule/japs_sync/hooks.php';
 require_once $CFG->dirroot.'/mod/mumiemodule/lib.php';

    /**
     * Function to handle the user_updated-event
     *
     * @param object $eventdata - the event's data
     */
    function user_updated_handler($eventdata){
        global $CFG;
        //check if the user is relevant for MUMIE
        if(record_exists('mumiemodule_students', 'userid', $eventdata->id)){
             if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
                fputs($LOGG, "user_updated_handler called \n");
                fputs($LOGG, dump_array($eventdata));
                fputs($LOGG, "\n \n");
             }
            $changeduser = new object();
            $changeduser->id = 'moodle-'.$CFG->prefix.'user-'.$eventdata->id;
            $changeduser->loginname = $eventdata->username;
            $changeduser->passwordencrypted = $eventdata->password;
            $changeduser->firstname = $eventdata->firstname;
            $changeduser->surname = $eventdata->lastname;
            $changeduser->matrnumber = 'X';
            change_user_for_mumie($changeduser);
        }
        return true;
    }

    /**
     * Function to handle the password_changed-event
     *
     * @param object $eventdata - the event's data
     */
    function password_changed_handler($eventdata){
        global $CFG;
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
                fputs($LOGG, "password_changed_handler called \n");
                fputs($LOGG, dump_array($eventdata));
                fputs($LOGG, "\n \n");
        }
        //check if the user is relevant for MUMIE
        if(record_exists('mumiemodule_students', 'userid',
            $eventdata->user->id)){
            $changeduser = get_mumie_user($eventdata->user);
```

```
        change_user_for_mumie($changeduser);
    }
    return true;
}
/**
 * Function to handle the user_deleted-event
 *
 * @param object $eventdata - the event's data
 */
function user_deleted_handler($eventdata){
    global $CFG;
    //check if the user is relevant for MUMIE
    if(record_exists('mumiemodule_students', 'userid', $eventdata->id)){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "user_deleted_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
        //TODO
        //insert functionality when sync-order is implemented in MUMIE
    }
    return true;
}

/**
 * Function to handle the course_updated-event
 *
 * @param object $eventdata - the event's data
 */
function course_updated_handler($eventdata){
    global $CFG;
    //check if the course is relevant
    if(record_exists('mumiemodule', 'course', $eventdata->id)){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "course_updated_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
        $class = new object();
        $class->syncid = 'moodle-'.$CFG->prefix.'course-'.$eventdata->id;
        $class->name = $eventdata->fullname;
        $class->description = $eventdata->summary;
        $class->semester = 'moodle-'.$CFG->prefix.
                           'course_categories-'.$eventdata->category;
        //perhaps the course has been put into another category,
        //so we better check if MUMIE knows
        $semester = new object();
        $semester->syncid = $eventdata->category;
        $semester->name = get_field('course_categories', 'name', 'id',
                          $eventdata->category);
        $semester->description = 'a Moodle-category as semester';
        if(!check_semester($semester)){
            $js = new JapsSynchronise();
            insert_semester_for_mumie($semester, $js);
            change_class_for_mumie($class, $js);
        } else{
            //lecturers can not be changed here
            change_class_for_mumie($class);
        }
```

```
    }
    return true;
}

/**
 * Function to handle the course_deleted-event
 *
 * @param object $eventdata - the event's data
 */
function course_deleted_handler($eventdata){
    global $CFG;
    //check if the course is relevant
    if(strpos($eventdata->modinfo, 'mumiemodule')!==false){
        //so there had been at least one mumiemodule in this course
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "course_deleted_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
        $classid = 'moodle-'.$CFG->prefix.'course-'.$eventdata->id;
        //TODO
        //delete_class_for_mumie($classid);
    }
    return true;
}

/**
 * Function to handle the category_updated-event
 *
 * @param object $eventdata - the event's data
 */
function category_updated_handler($eventdata){
    global $CFG;
    global $COURSE;
    global $USER;
    //check if this category is a MUMIE-semester
    $isinteresting = false;
    $modid = get_field('modules', 'id', 'name', 'mumiemodule');
    $modcourses = get_records('course_modules', 'module',
                              $modid, '', 'course');
    foreach($modcourses as $modcourse){
        if(record_exists('course', 'id', $modcourse->course,
                         'category', $eventdata->id)){
            $isinteresting = true;
            break;
        }
    }
    if($isinteresting){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "category_updated_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
        $semester = new object();
        $semester->syncid = 'moodle-'.
                          $CFG->prefix.'course_categories-'.$eventdata->id;
        $semester->name = $eventdata->name;
        $semester->description = 'a Moodle-category as semester';
        change_semester_for_mumie($semester);
```

```
    }
    return true;
}

/**
 * Function to handle the category_deleted-event
 *
 * @param object $eventdata - the event's data
 */
function category_deleted_handler($eventdata){
    global $CFG;
    $isinteresting = false; //we assume that this is not relevant - check!
    //if there is no class with an instance of mumiemodule in the
    //parent category, then we can be sure that this event isn't interesting
    //but if there is at least one, we can not be sure
    //so we better try to send all information
    if($cat_classes = get_courses($semester->parent, 'c.id')){
        foreach($cat_classes as $cat_class){
            if(record_exists('mumiemodule', 'course', $cat_class->id)){
                $isinteresting = true;
                break;
            }
        }
    }

    if($isinteresting){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "category_deleted_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
        $js = new JapsSynchronise();
        $deletedsem = new object();
        $deletedsem->syncid = 'moodle-'.
                        $CFG->prefix.'course_categories-'.$eventdata->id;
        $deletedsem->parent = 'moodle-'.
                      $CFG->prefix.'course_categories-'.$eventdata->parent;

        //we try to insert the parent-category as a new semester into MUMIE
        $cat = get_record('course_categories', 'id', $eventdata->parent, '',
                        '', '', '', 'name');
        $newsemester = new object();
        $newsemester->syncid = $deletedsem->parent;
        $newsemester->name = $cat->name;
        $newsemester->description = 'a Moodle-category as semester';
        insert_semester_for_mumie($newsemester, $js);
        //the courses were moved from the deleted into the parent category
        //so we have to change them before deleting the category
        if($cat_classes = get_courses($semester->parent, $fields='c.id')){
            foreach($cat_classes as $cat_class){
                if(record_exists('mumiemodule', 'course', $cat_class->id)){
                    $classtochange = new object();
                    $classtochange->syncid = 'moodle-'.
                                    $CFG->prefix.'course-'.$cat_class->id;
                    $classtochange->semester = $deletedsem->parent;
                    change_class_for_mumie($classtochange, $js);
                }
            }
        }
```

```
        //delete_semester_for_mumie($deletedsem);
    }
    return true;
}
/**
 * Function to handle the group_created-event
 *
 * @param object $eventdata - the event's data
 */
function group_created_handler($eventdata){
    global $CFG;
    //check if this group is a MUMIE tutorial
    if(record_exists('mumiemodule', 'course', $eventdata->courseid)){
        //later Moodle-Versions check if group fits activity-grouping
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "group_created_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
        send_new_tutorial_to_mumie($eventdata);
    }

    return true;
}

/**
 * Function to handle the group_updated-event
 *
 * @param object $eventdata - the event's data
 */
function group_updated_handler($eventdata){
    global $CFG;
    global $USER;
    global $COURSE;
    //check if this group is relevant
    if(record_exists('mumiemodule_students', 'groupid', $eventdata->id)){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "group_updated_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
        $tutorial = new object();
        $tutorial->id = 'moodle-'.$CFG->prefix.'groups-'.$eventdata->id;
        $tutorial->name = $eventdata->name;
        $tutorial->description = $eventdata->description;
        //tutors can not be changed here
        $tutorial->classid = 'moodle-'.$CFG->prefix.'course-'.
                             $eventdata->courseid;
        change_tutorial_for_mumie($tutorial);
    }

    return true;
}
```

```
/**
 * Function to handle the group_user_added-event
 *
 * @param object $eventdata - the event's data
 */
function group_user_added_handler($eventdata){
    global $CFG;
    //check if this group is relevant
    $courseid = groups_get_course($eventdata->groupid);
    if(record_exists('mumiemodule_students', 'groupid',
                          $eventdata->groupid)){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "group_user_added_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
         }
         //check if current user might be tutor or not
         $context = get_context_instance(CONTEXT_GROUP,
                    $eventdata->groupid);
         $group = groups_get_group($eventdata->groupid);
         if(has_capability('mod/mumiemodule:tutorize', $context,
                           $eventdata->userid)){ //this is a tutor
            //groups must not have more than one tutor in this activity!
            //REMOVE AFTER MUMIE-DB CHANGE
            $fields = 'u.id';
            $groupcontext = get_context_instance(CONTEXT_GROUP,
                                                 $eventdata->groupid);
            $grouptutors = get_users_by_capability($groupcontext,
                                                   'mod/mumiemodule:tutorize',
                                                   $fields, '', '', '',
                                                   $eventdata->groupid);
            if(count($grouptutors)>1){
                $removed = groups_remove_member($eventdata->groupid,
                                                $eventdata->userid);
                error(get_string('toomuchtutorssingle', 'mumiemodule'),
                                 "../group/assign.php?courseid=".$courseid."
                                 &group=".$eventdata->groupid.
                                 "&grouping=-1");
                break;
            } //REMOVE TILL HERE
            if(!has_capability('mod/mumiemodule:teachcourse',
                               $context, $eventdata->userid)){
                //so this is really just a tutor
                //lecturers are already known in mumie
                //perhaps the tutor is already known in MUMIE
                $tutortocheck = new object();
                $tutortocheck->syncid = 'moodle-'.$CFG.'user-'.
                                        $eventdata->userid;
                $tutorexists = check_tutor($tutortocheck);
                $islecturer = check_lecturer($tutortocheck);
                if(!$tutorexists && !$islecturer){
                    $newuser = get_record('user', 'id', $eventdata->userid);
                    send_single_user_to_mumie($newuser);
                }
            }
```

```
            //start creating the tutorial-object
            $tutorial = new object();
            $tutorial->syncid = 'moodle-'.$CFG->prefix.'groups-'.
                                $eventdata->groupid;
            $tutorial->name = $name->name;
            $tutorial->description = $group->description;
            $tutorial->tutor = 'moodle-'.$CFG->prefix.'user-'.
                               $eventdata->userid;
            $tutorial->classid = 'moodle-'.$CFG->prefix.'course-'.$courseid;
            change_tutorial_for_mumie($tutorial);
        } else if(has_capability('mod/mumiemodule:participate', $context,
                                 $eventdata->userid)){ //this is a student
            //a student must not be in more than one group in this activity
            $coursecontext = get_context_instance(CONTEXT_COURSE,
                                                  $courseid);
            $studentsgroups = groups_get_groups_for_user
                                        ($eventdata->userid, $courseid);
            if(count($studentsgroups)>1){
               $removed = groups_remove_member($eventdata->groupid,
                                               $eventdata->userid);
               error(get_string('justonegroupsingle', 'mumiemodule'),
                                "../group/assign.php?courseid=".$courseid."
                                 &group=".$eventdata->groupid.
                                 "&grouping=-1");
               break;
            }

            $mumiemodules = get_records('mumiemodule', 'course',
                                        $courseid);
            //keep in mind if there are more than one mumiemodules
            $morethanone = false;
            foreach($mumiemodules as $mumiemodule){
               mumiemodule_students_subscribe($eventdata->userid,
                                              $mumiemodule->id, $group,
                                              $morethanone);
               $morethanone = true;
            }
        }
    }
    return true;
}

/**
 * Function to handle the group_user_removed-event
 *
 * @param object $eventdata - the event's data
 */
function group_user_removed_handler($eventdata){
    global $CFG;
    //check if this group is relevant
    $courseid = groups_get_course($eventdata->groupid);
    if(record_exists('mumiemodule_students', 'groupid',
                     $eventdata->groupid)){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
            fputs($LOGG, "group_user_removed_handler called \n");
            fputs($LOGG, dump_array($eventdata));
            fputs($LOGG, "\n \n");
        }
```

```
        //check if removed user might be tutor or not
        $context = get_context_instance(CONTEXT_GROUP,
                                        $eventdata->groupid);
        if(has_capability('mod/mumiemodule:tutorize', $context,
                          $eventdata->userid)){
           //while MUMIE-DB is not changed we insert the Dummy-Tutor:
           $tutorial = new object();
           $tutorial->id = 'moodle-'.$CFG->prefix.'groups-'.
                           $eventdata->groupid;
           $tutorial->name = $eventdata->name;
           $tutorial->description = $group->description;
           $tutorial->tutor = 'moodle-dummy_tutor';
           $tutorial->classid = 'moodle-'.$CFG->prefix.'course-'.$courseid;
           change_tutorial_for_mumie($tutorial);
        } else if(has_capability('mod/mumiemodule:participate', $context,
                                 $eventdata->userid)){ //this is a student
           $userid = 'moodle-'.$CFG->prefix.'user-'.$eventdata->userid;
           $tutorialid = 'moodle-'.$CFG->prefix.'groups-'.
                         $eventdata->groupid;
           remove_user_from_mumie_tutorial($userid, $tutorialid);
           //a user must be part in exactly one tutorial per course
           //so the user has to be removed from the modules of this course
           $mumiemodules = get_records('mumiemodule', 'course', $courseid);
           foreach($mumiemodules as $mumiemodule){
               mumiemodule_students_unsubscribe($eventdata->userid,
                                                $mumiemodule->id,
                                                $eventdata->groupid);
           }
        }
    }
    return true;
}


/**
 * Function to handle the group_deleted-event
 *
 * @param object $eventdata - the event's data
 */
function group_deleted_handler($eventdata){
    global $CFG;
    global $COURSE;
    //check if this group is relevant
    if(record_exists('mumiemodule_students', 'groupid', $eventdata->group)){
        if($LOGG = fopen($CFG->dirroot.'/mumie_debug/event_log.txt', 'a')){
           fputs($LOGG, "group_deleted_handler called \n");
           fputs($LOGG, dump_array($eventdata));
           fputs($LOGG, "\n \n");
        }
        $tutorialid = 'moodle-'.$CFG->prefix.'groups-'.$eventdata->group;

        //if there had been students in that group, remove
        //THIS PART OF CODE SHOULD BE REMOVED
        //WHEN THE DELETE-SYNC-ORDER IS IMPLEMENTED IN MUMIE
        //MUMIE ITSELF SHOULD HANDLE THOSE TUTORIAL-INTERNAL REMOVEMENTS
        $groupusers = get_records('mumiemodule_students', 'group',
                                  $eventdata->group, '', 'userid');
        $js = new JapsSynchronise();
        $fields = 'id, username, password, firstname, lastname';
```

```
            foreach($groupusers as $groupuser){
                $userid = 'moodle-'.$CFG->prefix.'user-'.$groupuser->id;
                remove_user_from_mumie_tutorial($userid, $tutorialid, $js);
             }//END OF CODE TO REMOVE

            //delete_tutorial_for_mumie($tutorialid); //TODO
        }
        return true;
    }

?>
```

A1.11 Quellcode transfer_objects.php

```
<?php
/**
 * Library of classes and functions
 * for module mumiemodule
 *
 * @author PR
 * @package mumiemodule
 **/


class Semester {
    var $syncid = "";
    var $name = "";
    var $description = "";

   function __construct($syncid, $name, $description) {
       $this->syncid = $syncid;
       $this->name = $name;
       $this->description = $description;
   }

    function setDescription($description) {
       $this->description = $description;
   }

   function setSyncId($syncid) {
       $this->syncid = $syncid;
   }
}

class MumieClass {
    var $syncid = "";
    var $name = "";
    var $description = "";
    var $semester = null;
    var $lecturers = null;
    var $tutorials = null;
    var $mumie_course_id = "";

   function __construct($syncid, $name, $description, $semester, $lecturers,
                        $tutorials, $mumie_course_id) {
       $this->syncid = $syncid;
       $this->name = $name;
       $this->description = $description;
       $this->semester = $semester;
       $this->lecturers = $lecturers;
       $this->tutorials = $tutorials;
       $this->mumie_course_id = $mumie_course_id;
   }

   function setDescription($description) {
       $this->description = $description;
   }

   function setSyncId($syncid) {
       $this->syncid = $syncid;
```

```
    }

    function addLecturer($lecturer) {
        array_push($this->lecturers, $lecturer);
    }

    function setSemester($semester){
        $this->semester = $semester;
    }

    function addTutorial($tutorial) {
        array_push($this->tutorials, $tutorial);
    }

    function setMumieCourseId($mumie_course_id){
        $this->mumie_course_id = $mumie_course_id;
    }
}

class Tutorial {
    var $syncid = "";
    var $name = "";
    var $description = "";
    var $tutor = null;
    var $students = null;

    function __construct($syncid, $name, $description, $tutor, $students) {
        $this->syncid = $syncid;
        $this->name = $name;
        $this->description = $description;
        $this->tutor = $tutor;
        $this->students = $students;
    }

    function setDescription($description) {
        $this->description = $description;
    }

    function setSyncId($syncid) {
        $this->syncid = $syncid;
    }

     function setTutor($tutor) {
        $this->tutor = $tutor;
    }

     function addStudent($student) {
        array_push($this->students, $student);
    }
}

class User {
    var $syncid = "";
    var $loginname = "";
    var $passwordencrypted = "";
    var $firstname = "";
    var $surname = "";
    var $matrnumber = "X";
```

```
    function __construct($syncid, $loginname, $passwordencrypted, $firstname,
                         $surname, $matrnumber) {
        $this->syncid = $syncid;
        $this->loginname = $loginname;
        $this->passwordencrypted = $passwordencrypted;
        $this->firstname = $firstname;
        $this->surname = $surname;
        $this->matrnumber = $matrnumber;
    }

    function setpasswordencrypted($passwordencrypted){
        $this->passwordencrypted = $passwordencrypted;
    }

    function setFirstname($firstname){
        $this->firstname = $firstname;
    }

    function setSurname($surname){
        $this->surname = $surname;
    }

    function setMatrnumber($matrnumber){
        $this->matrnumber = $matrnumber;
    }

     function getName() {
        return $this->name;
    }
}

class Lecturer extends User {

    function __construct($syncid, $loginname, $passwordencrypted, $firstname,
                         $surname, $matrnumber) {
        parent::__construct($syncid, $loginname, $passwordencrypted, $firstname,
                            $surname, $matrnumber);
    }

}

class Tutor extends User {
    function __construct($syncid, $loginname, $passwordencrypted, $firstname,
                         $surname, $matrnumber) {
        parent::__construct($syncid, $loginname, $passwordencrypted, $firstname,
                            $surname, $matrnumber);
    }
}

class Student extends User {
    function __construct($syncid, $loginname, $passwordencrypted, $firstname,
                         $surname, $matrnumber) {
        parent::__construct($syncid, $loginname, $passwordencrypted, $firstname,
                            $surname, $matrnumber);
    }
}
?>
```

A1.12 Quellcode hooks.php

```
<?php
/**
 * Library of functions to send information to MUMIE
 * for module mumiemodule
 *
 * @author PR
 * @package mumiemodule
 **/

require_once 'HttpURLConnection.class.php';
require_once 'JapsClient.class.php';
require_once 'JapsSynchronise.class.php';

 global $CFG;
 global $COURSE;
 global $USER;
 require_once $CFG->dirroot.'/mod/mumiemodule/lib.php';
 require_once $CFG->dirroot.'/mod/mumiemodule/internalchecks.php';

 /**
  * Function to insert all neccesarry information out of a new MUMIE-module into
  * MUMIE
  *
  * @param object $insert - big object with all information
  */
  function insert_all_for_mumie($insert){
    global $CFG;
    global $COURSE;
    $js = new JapsSynchronise();
    //first we insert the new semester if MUMIE does not know it yet
    $semesterexists = check_semester($insert->semester);
    if(!$semesterexists){
        insert_semester_for_mumie($insert->semester, $js);
    }
    //now we insert the new class
    $newclass = new object();
    $newclass->syncid = $insert->syncid;
    $newclass->name = $insert->name;
    $newclass->description = $insert->description;
    $newclass->semester = $insert->semester->syncid;
    $newclass->lecturers = '';
    //let's be sure to have all lecturers as users in the MUMIE DB
    if($insert->lecturers){
        foreach($insert->lecturers as $lecturer){
            //perhaps the lecturer is already in the MUMIE
            $lecturerexists = check_lecturer($lecturer);
            if(!$lecturerexists){
                insert_user_for_mumie($lecturer, $js);
            }
            $newclass->lecturers .= $lecturer->syncid.' ';
        }
    }
    $newclass->mumie_course_id = $insert->mumie_course_id;
    insert_class_for_mumie($newclass, $js);
    //next is the tutorials
    if($insert->tutorials){
```

```
        foreach($insert->tutorials as $tutorial){
            //we should make sure that the tutor is a user in the Mumie DB
            if($tutorial->tutor){
                //perhaps the tutor is already in the MUMIE
                $tutorexists = check_tutor($tutorial->tutor);
                $tutor_is_lecturer = check_lecturer($tutorial->tutor);
                if(!$tutorexists && !$tutor_is_lecturer){
                    insert_user_for_mumie($tutorial->tutor, $js);
                }
                $tutorial->tutor = $tutorial->tutor->syncid;
            }
            else { //no tutor => insert dummy
                $tutorial->tutor = 'moodle-dummy_tutor';
            }
            $tutorial->classid = $insert->syncid;
            insert_tutorial_for_mumie($tutorial, $js);
            if($tutorial->students){
                foreach($tutorial->students as $student){
                    //perhaps MUMIE already knows this student so we first check
                    $studentexists = check_student($student);
                    if(!$studentexists){
                        insert_user_for_mumie($student, $js);
                    }
                    add_user_to_mumie_tutorial($student->syncid,
                                               $tutorial->syncid, $js);
                }
            }
        }
    }
 }


/**
 * Function to insert a Moodle-course_category into MUMIE as a semester
 *
 * @param $semester - object with specific information
 */
function insert_semester_for_mumie($semester, $js=null){
   $syncID = $semester->syncid;
   $syncData = array("sync-id"=>$syncID, "name"=>$semester->name,
                     "description"=>$semester->description);
   //send new semester info to MUMIE
   $sync_order = 'new-semester';
   if(!$js){
       $js = new JapsSynchronise();
   }
   // adding "protected/sync/" before the synchronisations type specific path
   $cocoonPath = "protected/sync/".$sync_order;
   // Starting the syncronisation and getting the result
   $response = $js->post($cocoonPath, $syncData);
   $filteredResponse = filter_response($response);
   logoutput($filteredResponse, $sync_order, $syncData);
   // Take a look if the response is okay
   if($filteredResponse != "OK"){
       //events API should care about errors
       //so no error-handling here - perhaps later
    }
}
```

```
/**
 * Function to change semester-information in MUMIE
 *
 * @param $semester - object containing the new information
 */
function change_semester_for_mumie($semester, $js=null){
   $syncID = $semester->syncid;
   $syncData = array("sync-id"=>$syncID, "name"=>$semester->name,
                     "description"=>$semester->description);
   $sync_order = 'change-semester-data';
   if(!$js){
       $js = new JapsSynchronise();
   }
   // adding "protected/sync/" before the synchronisations type specific path
   $cocoonPath = "protected/sync/".$sync_order;
   // Starting the syncronisation and getting the result
   $response = $js->post($cocoonPath, $syncData);
   $filteredResponse = filter_response($response);
   logoutput($filteredResponse, $sync_order, $syncData);
   // Take a look if the response is okay
   if($filteredResponse != "OK"){
       //space for error-handling if we need some later
    }
}

/**
 * Function to delete a semester in MUMIE and eventually
 * change it's classes to a new one
 *
 * @param object $semester - object containing the information
 * about deleted and parent semester
 */
 function delete_semester_for_mumie($semester, $js=null){
   global $CFG;
   if(!$js){
       $js = new JapsSynchronise();
   }
   //sync-order not implemented yet in MUMIE //TODO
 }

/**
 * Function to insert a Moodle-course into MUMIE as a new MUMIE-class
 *
 * @param $class - object containing the new class-information
 */
function insert_class_for_mumie($class, $js=null){
   $syncID = $class->syncid;
   $syncData = array("sync-id"=>$syncID, "name"=>$class->name,
                      "description"=>$class->description,
                     "semester"=>$class->semester,
                     "lecturers"=>$class->lecturers,
                     "courses"=>$class->mumie_course_id);
   //send new class to MUMIE
   $sync_order = 'new-class';
   if(!$js){
       $js = new JapsSynchronise();
   }
   // adding "protected/sync/" before the synchronisations type specific path
   $cocoonPath = "protected/sync/".$sync_order;
```

```
   // Starting the syncronisation and getting the result
   $response = $js->post($cocoonPath, $syncData);
   $filteredResponse = filter_response($response);
   logoutput($filteredResponse, $sync_order, $syncData);
   // Take a look if the response is okay
   if($filteredResponse != "OK"){
       //MUMIE error
    }
}

/**
 * Function to change class-information in MUMIE
 *
 * @param $class - object containing the changed information
 */
function change_class_for_mumie($class, $js=null){
   $syncID = $class->syncid;
   $syncData = array("sync-id"=>$syncID);
   if($class->name){
       $syncData["name"]=$class->name;
   }
   if($class->description){
       $syncData["description"]=$class->description;
   }
   if($class->semester){
       $syncData["semester"]=$class->semester;
   }
   if($class->lecturers){
       $syncData["lecturers"]=$class->lecturers;
   }
   //no change of MUMIE-course possible, so we discard
   $sync_order = 'change-class-data';
   if(!$js){
       $js = new JapsSynchronise();
   }
   // adding "protected/sync/" before the synchronisations type specific path
   $cocoonPath = "protected/sync/".$sync_order;
   // Starting the syncronisation and getting the result
   $response = $js->post($cocoonPath, $syncData);
   $filteredResponse = filter_response($response);
   logoutput($filteredResponse, $sync_order, $syncData);
   // Take a look if the response is okay
   if($filteredResponse != "OK"){
       //for later error-handling
    }
}

/**
 * Function to insert a Moodle-group into MUMIE as a new tutorial
 *
 * @param $tutorial - object containing the necessary information
 */
function insert_tutorial_for_mumie($tutorial, $js=null){
   $syncID = $tutorial->syncid;
   $syncData = array("sync-id"=>$syncID, "name"=>$tutorial->name,
                       "description"=>$tutorial->description,
                     "tutor"=>$tutorial->tutor, "class"=>$tutorial->classid);
   $sync_order = 'new-tutorial';
```

```
    if(!$js){
        $js = new JapsSynchronise();
    }
    // adding "protected/sync/" before the synchronisations type specific path
    $cocoonPath = "protected/sync/".$sync_order;
    // Starting the syncronisation and getting the result
    $response = $js->post($cocoonPath, $syncData);
    $filteredResponse = filter_response($response);
    logoutput($filteredResponse, $sync_order, $syncData);
    // Take a look if the response is okay
    if($filteredResponse != "OK"){
        //perhaps the tutorial already exists and we just want to update
        //change_tutorial_for_mumie($tutorial);
     }
}

/**
 * Function to change tutorial-information in MUMIE
 *
 * @param $tutorial - object containing the changed information
 */
function change_tutorial_for_mumie($tutorial, $js=null){
    $syncID = $tutorial->syncid;
    $syncData = array("sync-id"=>$syncID);
    if($tutorial->name){
        $syncData["name"]=$tutorial->name;
    }
    if($tutorial->description){
        $syncData["description"]=$tutorial->description;
    }
    //if($tutorial->tutor){
        $syncData["tutor"]=$tutorial->tutor;
    //}
    if($tutorial->classid){
        $syncData["class"]=$tutorial->classid;
    }
    $sync_order = 'change-tutorial-data';
    if(!$js){
        $js = new JapsSynchronise();
    }
    // adding "protected/sync/" before the synchronisations type specific path
    $cocoonPath = "protected/sync/".$sync_order;
    // Starting the syncronisation and getting the result
    $response = $js->post($cocoonPath, $syncData);
    $filteredResponse = filter_response($response);
    logoutput($filteredResponse, $sync_order, $syncData);
    // Take a look if the response is okay
    if($filteredResponse != "OK"){
        //for later error-handling
     }
}
```

```
/**
 * Function to insert a new user into MUMIE
 *
 * @param $user - object containing the information about the new user
 */
   function insert_user_for_mumie($newuser, $js=null){
       //the Moodle_to_MUMIE_login_workaround:
       $newuser->passwordencrypted = hash_internal_user_password
                                                    ($newuser->passwordencrypted);
       $syncID = $newuser->syncid;
       $syncData = array("sync-id"=>$syncID,"login-name"=>$newuser->loginname,
                         "password-encrypted"=>$newuser->passwordencrypted,
                         "first-name"=>$newuser->firstname,
                          "surname"=>$newuser->surname,
                         "matr-number"=>$newuser->matrnumber);
       $sync_order = 'new-user';
       if(!$js){
           $js = new JapsSynchronise();
       }
       // adding "protected/sync/" before the synchronisations
       //type specific path
       $cocoonPath = "protected/sync/".$sync_order;
       // Starting the syncronisation and getting the result
       $response = $js->post($cocoonPath, $syncData);
       $filteredResponse = filter_response($response);
       logoutput($filteredResponse, $sync_order, $syncData);
       // Take a look if the response is okay
       if($filteredResponse != "OK"){
           //for later error-handling
       }
    }

    /**
     * Function to change user-information in MUMIE+
     *
     * @param $newUser - object with the changed user-data
     */
    function change_user_for_mumie($newuser, $js=null){
       //Moodle to MUMIE login_workaround:
       $newuser->passwordencrypted = hash_internal_user_password
                                                    ($newuser->passwordencrypted);
       $syncID = $newuser->syncid;
       $syncData = array("sync-id"=>$syncID,"login-name"=>$newuser->loginname,
                         "password-encrypted"=>$newuser->passwordencrypted,
                         "first-name"=>$newuser->firstname,
                         "surname"=>$newuser->surname,
                         "matr-number"=>$newuser->matrnumber);
       $sync_order = 'change-user-data';
       if(!$js){
           $js = new JapsSynchronise();
       }
       // adding "protected/sync/" before the synchronisations type
       //specific path
       $cocoonPath = "protected/sync/".$sync_order;
       // Starting the syncronisation and getting the result
       $response = $js->post($cocoonPath, $syncData);
       $filteredResponse = filter_response($response);
       logoutput($filteredResponse, $sync_order, $syncData);
```

```
        // Take a look if the response is okay
        if($filteredResponse != "OK"){
            //for later error-handling
        }
     }

/**
 * Function to add a Moodle-group-user into a MUMIE-tutorial
 *
 * @param $newuserID - string containing the Sync-ID of the user for MUMIE
 * @param $syncID - string containing the Sync-ID of the tutorial for MUMIE
 */
   function add_user_to_mumie_tutorial($newuserID, $syncID, $js=null){
        $syncData = array("tutorial"=>$syncID, "user"=>$newuserID);
        $sync_order = 'add-user-to-tutorial';
        if(!$js){
            $js = new JapsSynchronise();
        }
        // adding "protected/sync/" before the synchronisations type
        //specific path
        $cocoonPath = "protected/sync/".$sync_order;
        // Starting the syncronisation and getting the result
        $response = $js->post($cocoonPath, $syncData);
        $filteredResponse = filter_response($response);
        logoutput($filteredResponse, $sync_order, $syncData);
        // Take a look if the response is okay
        if($filteredResponse != "OK"){
            //error-handling
        }
   }

/**
 * Function to remove a Moodle-group-user from a MUMIE-tutorial
 *
 * @param $newuserID - string containing the Sync-ID of the user for MUMIE
 * @param $syncID - string containing the Sync-ID of the tutorial for MUMIE
 */
   function remove_user_from_mumie_tutorial($userID, $syncID, $js=null){
        $syncData = array("tutorial"=>$syncID, "user"=>$userID);
        $sync_order = 'remove-user-from-tutorial';
        if(!$js){
            $js = new JapsSynchronise();
        }
        // adding "protected/sync/" before the synchronisations type
        //specific path
        $cocoonPath = "protected/sync/".$sync_order;
        // Starting the syncronisation and getting the result
        $response = $js->post($cocoonPath, $syncData);
        $filteredResponse = filter_response($response);
        logoutput($filteredResponse, $sync_order, $syncData);
        // Take a look if the response is okay
        if($filteredResponse != "OK"){
            //for error-handling later
        }
   }
```

```
/*******************************************************************************
 *                            Getter Functions
 ******************************************************************************/

 /**
  * Function to get a list of all courses in MUMIE
  *
  * @param $js - instance of class JapsSynchronise (optional)
  */
 function get_mumie_courses($js = null){
    //$get_order = 'get-courses';
    if(!$js){
        $js = new JapsSynchronise();
    }
    $cocoonPath = "protected/data/document-index/type-name/course";
    // Starting the request and getting the result
    $response = $js->get($cocoonPath);
    $filteredResponse = filter_response($response);
    $course_array = filter_response_to_array($filteredResponse, 'courses');

    //if we do not want to also have the courses that are already in use,
    //we put that code back in
    /*
    $return_array = array();
    foreach($course_array as $course_item){
        if($course_item->alreadyused === false){
            array_push($return_array, $course_item);
        }
    }
    return $return_array;
    */
    return $course_array;
 }

 /**
  * Function to get a list with the current points for all students of one class
  */
  function get_mumie_grades($courseid, $js = null){
    global $CFG;
    if(!$js){
        $js = new JapsSynchronise();
    }
    //workaround - we do not want to be redirected because not logged in:
    $login_response = $js->login();

    $cocoonPath = "protected/data/total-class-grades";
    $class_sync_id = 'moodle-'.$CFG->prefix.'course-'.$courseid;
    // Starting the syncronisation and getting the result
    $response = $js->get($cocoonPath.'?sync-id='.$class_sync_id);
    $filteredResponse = filter_response($response);
    $grade_array = filter_response_to_array($filteredResponse, 'points');
    return $grade_array;
  }
```

```
/*****************************************************************************
 *                                Help Functions
 *****************************************************************************/

/**
  * Function to filter an answer of JapsSynchronise to string
  *
  * @param $array - object of the unfiltered array
  */
    function filter_response($array) {
   $result = "";
   if(is_array($array) || is_object($array)){
    foreach($array as $key => $value) {
        if($key == "body"){
                return $value;
            }
        }
        if(is_array($value) || is_object($value)) {
            dump_array($value);
        }
    }
    else {
        return "error";
        //return var_dump($array);
    }
}

/**
 * Function to get an array as a list out of a MUMIE response
 */
 function filter_response_to_array($response, $mode) {
   global $CFG;
   $xml = simplexml_load_string($response);
   $array = array();
   switch($mode){
   case 'courses':
       foreach ($xml->children('http://www.mumie.net/xml-
                  namespace/document/metainfo') as $item) {
          $course_attributes = $item->attributes();
          $course_id = (int)$course_attributes['id'];
          $newcourse = new object();
          $newcourse->id = $course_id;
          $newcourse->name = (string)$item->name;
          if($item->class){
            $newcourse->alreadyused = true;
          } else{
            $newcourse->alreadyused = false;
          }
          array_push($array, $newcourse);
        }
        break;
   case 'points':
        $iterator = 0;
        $user_sync_id_prefix = 'moodle-'.$CFG->prefix.'user-';
        $user_sync_id_prefix_length = strlen($user_sync_id_prefix);
        foreach ($xml->children('http://www.mumie.net/xml-namespace/grades')
                  as $item) {
            $itemlength = count($xml->children('http://www.mumie.net/xml-
                                                 namespace/grades'));
```

```
            ++$iterator;
            if($iterator<$itemlength){
                $grade_attributes = $item->attributes();
                $user_sync_id = (string)$grade_attributes['sync_id'];
                $new_user_grade = new object();
                $new_user_grade->userid = substr($user_sync_id,
                                                    $user_sync_id_prefix_length);
                $new_user_grade->current_points =
                                            (double)$grade_attributes['points'];
                $new_user_grade->edited_problems =
                                        (int)$grade_attributes['edited_problems'];
                $new_user_grade->corrected_problems =
                                    (int)$grade_attributes['corrected_problems'];
                array_push($array, $new_user_grade);
            } elseif($iterator==$itemlength){
                $grade_attributes = $item->attributes();
                $mumie_course = new object();
                $mumie_course->course_name =
                                        (string)$grade_attributes['course_name'];
                $mumie_course->total_problems =
                                         (int)$grade_attributes['total_problems'];
                $mumie_course->total_points =
                                         (int)$grade_attributes['total_points'];
                array_push($array, $mumie_course);
            }
        }
        break;
   }
    return $array;
}

/**
 * Function to var_dump an array
 *
 * @param $array - array to be dumped out
 */
function dump_array($array) {
   $result = "";
   if(is_array($array) || is_object($array)){
    foreach($array as $key => $value) {
        if(is_array($value) || is_object($value)) {
            //$result .= "Array:$key =>" . dump_array($array);
        }
        else {
            $result .= "\$key:$key = \$value:$value\n";
        }
    }
    return $result;
    }
    else {
        return "";
    }

}
```

```
/**
 * Function to write some information into a logfile
 */
function logoutput($filteredResponse, $order, $syncData){
    global $CFG;
    if($LOGG = fopen($CFG->dirroot.'/mod/mumiemodule/logs/hook_log.txt', 'a')){
        fputs($LOGG, $filteredResponse." bei ".$order);
        /*if($filteredResponse != "OK"){
            fputs($LOGG, '\n'."SyncData war: ");
            fputs($LOGG, dump_array($syncData));
        }*/
        fputs($LOGG, "\n");
    }
}

/**
 * yet another function to write information and the current time into logfile
 */
function logout_time($nachricht=''){
    global $CFG;
    if($LOGG = fopen($CFG->dirroot.'/mod/mumiemodule/logs/hook_log.txt', 'a')){
        fputs($LOGG, $nachricht);
        fputs($LOGG, "aktuelle Zeit: ".date("l dS of F Y h:i:s A"));
        fputs($LOGG, "\n");
    }
}


?>
```

A1.13 Quellcode JapsClient.class.php

```
<?php


abstract class JapsClient
{

  //
----------------------------------------------------------------------------
  // Static constants
  //
----------------------------------------------------------------------------

  /**
   * Name of the response header indicating that the user must login
   * (<code>"X-Mumie-Login-Required"</code>).
   */
   const LOGIN_REQUIRED_HEADER = "X-Mumie-Login-Required";

  /**
   * Indicates a successful login.
   */
   const LOGIN_SUCCESSFUL = 0;

  /**
   * Indicates a failed login.
   */
   const LOGIN_FAILED = 1;

  /**
   * Indicates that the uses has canceled the login dialog.
   */
   const LOGIN_CANCELED = 2;

  /**
   * The url path for login (<code>"login"</code>).
   * Together with {@link #urlPrefix}, this
   * is the url where to send the login form data.
   */
   const LOGIN_PATH = "public/auth/login";

  /**
   * The request parameter containing the account (<code>"name"</code>).
   */
   const ACCOUNT_PARAM = "name";

  /**
   * The request parameter containing the password (<code>"password"</code>).
   */
   const PASSWORD_PARAM = "password";

  /**
   * The request parameter containing the resource (<code>"resource"</code>).
   */
   const RESOURCE_PARAM = "resource";
```

```
  /**
   * The url path of the resource requested in the login request
   * (<code>"protected/auth/login-successful"</code>).
   */
   const RESOURCE_VALUE = "protected/auth/login-successful";

  /**
   * Unlimited number of login tries. Value: <code>-1</code>
   */
   const UNLIMITED = -1;

  /**
   * Content type suited for text files (<code>"text/plain"</code>).
   */
   const TEXT_CONTENT_TYPE = "text/plain";

  /**
   * Content type suited for binary files
   * (<code>"application/octet-stream"</code>).
   */
   const BINARY_CONTENT_TYPE = "application/octet-stream";

  /**
   * Default content type for file uploads ({@link #BINARY_CONTENT_TYPE}).
   */
   const DEFAULT_CONTENT_TYPE = "BINARY_CONTENT_TYPE";

  /**
   * The common prefix of the log messages (<code>"JapsClient"</code>).
   */
   const LOG_PREFIX = "JapsClient";

  /**
   * The default timestamp pattern (<code>"yyyy-MM-dd HH:mm:ss S"</code>)
   */
   const DEFAULT_TIMESTAMP_PATTERN = "Y-m-d H:i:s";


  //
------------------------------------------------------------------------------
  // Global variables
  //
------------------------------------------------------------------------------

  /**
   * The url prefix of the Japs server.
   */
   protected $urlPrefix = "";

  /**
   * The url prefix of the Japs server without Port.
   */
   protected $urlPrefixWithoutPort = "";

  /**
   * The Port for the connection. Default: Port 80.
   */
   protected $urlPort = 80;
```

```
/**
 * Contains the String "cocoon/" to add to the path
 */
 protected $cocoon = "";

/**
 * Maximum number of login tries. A value of {@link #UNLIMITED}
 * (<code>-1</code>) means the
 * number is not limited. Default: {@link #UNLIMITED}.
 */
 protected $maxLoginTries = self::UNLIMITED;

/**
 * <p>
 *   Stores content types (i.e., MIME types) for several file suffixes.
 *   The default is as
 *   follows:
 * </p>
 * <table class="genuine">
 *   <thead>
 *     <tr><td>Suffix</td><td>Content Type</td></tr>
 *   </thead>
 *   <tbody>
 *     <tr><td><code>txt</code></td><td><code>text/plain</code></td></tr>
 *     <tr><td><code>xml</code></td><td><code>text/xml</code></td></tr>
 *     <tr><td><code>xhtml</code></td><td><code>text/xml</code></td></tr>
 *     <tr><td><code>xsl</code></td><td><code>text/xsl</code></td></tr>
 *     <tr><td><code>html</code></td><td><code>text/html</code></td></tr>
 *     <tr><td><code>css</code></td><td><code>text/css</code></td></tr>
 *     <tr><td><code>tex</code></td><td><code>text/tex</code></td></tr>
 *     <tr><td><code>jpg</code></td><td><code>image/jpeg</code></td></tr>
 *     <tr><td><code>png</code></td><td><code>image/png</code></td></tr>
 *     <tr><td><code>zip</code></td><td><code>application/zip</code></td></tr>
 *     <tr><td><code>jar</code></td><td><code>application/x-java-
 *      archive</code></td></tr>
 *   </tbody>
 * </table>
 */
 protected $contentTypes = array();

/**
 * Save the filename, when a logfile is specified.
 */
 protected $logFile = "";

/**
 * Save received cookies.
 */
 protected $cookies = array();
```

```
  //
-------------------------------------------------------------------------------
  // Constructor
  //
-------------------------------------------------------------------------------

  /**
   * <p>
   *   Creates a new Japs client with <code>urlPrefix</code>
   *    as server url prefix and <code>filename</code> as a logfile.
   * </p>
   * <p>
   *   Extending classes should call this in their constructors.
   * </p>
   */

  public function JapsClient ($urlPrefix, $logFile = "")
  {
    // URL prefix:
    if ($urlPrefix == "" || $urlPrefix == "/" )
    {  throw new Exception("Url prefix null or void");  }
    $lastIndex = strlen($urlPrefix) -1;
    if ( substr($urlPrefix,$lastIndex,$lastIndex) == "/" )
    {  $urlPrefix = substr($urlPrefix,0,$lastIndex);  }
    $urlPrefixArray = explode("/",$urlPrefix);
    if(!strpos($urlPrefixArray[2],":"))
    {  $this->urlPrefixWithoutPort = $urlPrefixArray[2];  }
    elseif(strpos($urlPrefixArray[2],":"))
    {  $helparr = explode(":",$urlPrefixArray[2]);
       $this->urlPrefixWithoutPort = $helparr[0];
       $this->urlPort = $helparr[1];
    }
    if(array_key_exists(3,$urlPrefixArray))
    {  $this->cocoon = $urlPrefixArray[3].'/';  }

    $this->urlPrefix = $urlPrefix;

    $this->logFile = $logFile;
    if(file_exists($this->logFile))
    {  unlink($this->logFile);  }



    // Content types table:
    $this->contentTypes["txt"] = "text/plain";
    $this->contentTypes["xml"] = "text/xml";
    $this->contentTypes["xhtml"] = "text/xml";
    $this->contentTypes["xsl"] = "text/xsl";
    $this->contentTypes["html"] = "text/html";
    $this->contentTypes["css"] = "text/css";
    $this->contentTypes["tex"] = "text/tex";
    $this->contentTypes["jpg"] = "image/jpeg";
    $this->contentTypes["png"] = "image/png";
    $this->contentTypes["zip"] = "application/zip";
    $this->contentTypes["jar"] = "application/x-java-archive";
  }
```

```
  //
-------------------------------------------------------------------------------
  // Hooks for cookie handling
  //
-------------------------------------------------------------------------------

  /**
   * <p>
   *   If necessary, reads the cookies revieved from the server
   *   and stores them in cookies.
   * </p>
   */
   protected function handleRecievedCookies($connect)
      //throws JapsClientException
   {
      if(!array_key_exists('cookies',$connect['head']))
      {
         //echo "no cookie to save";
      }
      else
      {
         $this->cookies = $connect['head']['cookies'];
      }
   }

  /**
   * <p>
   *   If necessary, sets the cookies that should be sent back to the server.
   * </p>
   */

   protected function addCookies () //throws JapsClientException
   {
      $cookieString = "";

      if(empty($this->cookies))
      {
         return false;
      }
      else
      {
         foreach($this->cookies as $cookie)
         {
            if(is_array($cookie))
            {
               if(!array_key_exists('name',$cookie)
                   || !array_key_exists('value',$cookie))
               { echo "Bad Cookie saved!!";  }
               else
               {
                  $cookieString .= urlencode($cookie['name'])
                                   .'='.urlencode($cookie['value']).';';
               }
            }
         }
      }
      return $cookieString;
  }
```

```
  //
-------------------------------------------------------------------------------
  // HTTP Get and Post requests
  //
-------------------------------------------------------------------------------

  /**
   * <p>
   *   Returns a connection for <code>path</code> via the HTTP Get method.
   * </p>
   * <p>
   *   The entries of <code>params</code> are added
   *   as request parameters to the URL. They
   *   are automatically url-encoded. If <code>params</code>
   *   is <code>null</code> or empty,
   *   no request parameters are added.
   * </p>
   * <p>
   *   If <code>loginIfNecessary</code> is <code>true</code>,
   *   the user is logged-in if the
   *   server requires so. This is done by {@link #login()}
   *   (which normally allowes several
   *   tries to log-in). If the login succeded, the request is repeated.
   *   If the server
   *   requires login again (which should not happen in normal operation),
   *   an exception is
   *   thrown. If the login failed, an exception is thrown.
   *   If the login was canceled by the
   *   user, no exception is thrown and <code>null</code> is returned.
   * </p>
   */
/*=========================================*\
                                      GET
\*=========================================*/

  public function get($path, $params, $loginIfNecessary)
    //throws JapsClientException, IOException, MalformedURLException
  {
     $numargs = func_num_args();
     if($numargs == 1)   //get($path)
     {
        $path = func_get_arg(0);
        return $this->get($path,$params = array());
     }
     elseif($numargs == 2)  //get($path, $params)
     {
        $path = func_get_arg(0);
        $params = func_get_arg(1);
        return $this->get($path,$params, true);
     }
     elseif($numargs == 3)  //get($path, $params, $loginIfNecessary)
     {
        $path = func_get_arg(0);
        $params = func_get_arg(1);
        $loginIfNecessary = func_get_arg(2);

        $newpath = $this->cocoon.$path; // add "cocoon/" to path

        $METHOD = "get(String,Array,Boolean)";
```

```
  $this->logg($METHOD." 1/2: path = ".$newpath.", params = "
              .$this->paramsToString($params).", loginIfNecessary = ");
  $this->logg($loginIfNecessary ? 'true<br/>' : 'false<br/>');

  // Prepare connection:
 $connection = new HttpURLConnection($this->urlPrefixWithoutPort,
                                     $this->urlPort,
                                     false,"JapsClient/PHP");
 // Prepare params
 $params = $this->url_Encode($params);
 // Connect and add Cookies:
 $connect = $connection->get($newpath,$params,$this->addCookies());
 //$this->logg(var_dump($connect));
 // Check response code:
 if( $connect["head"]["code"] >= 400)  // Error
 {  echo "Could not connect";
     return; //sonst wird login-Abschnitt ausgefuehrt!!!
 }

 // If necessary, handle cookies:
 $this->handleRecievedCookies($connect);

 // Redirect if necessary:
 if (  $connect["head"]["code"] >= 300 && $connect["head"]["code"] < 400)
// Redirected
 {
    // Follow redirect (by Http Get):
    $location = $connect["head"]["location"]["uri"];
    $params = $connect["head"]["location"]["parameters"];
    $connect = $this->get($this->getPath($location), $params, false);

    // Check response code again; fail if not OK:
    if( $connect["head"]["code"] != 200)  // Code 200: HTTP_OK?
    {  echo "Could not connect";
       return; //sonst wird login-Abschnitt ausgefuehrt!!!
    }
 }

 // Login if necessary:
 if ( $loginIfNecessary && $this->checkLoginRequired($connect))
 {
    $loginStatus = $this->login();

    switch ( $loginStatus )
    {
       case self::LOGIN_SUCCESSFUL:
       {
          $connect = $this->get($path, $params, false);
          if ($this->checkLoginRequired($connect))
            echo "rejected after successful login";
          break;
       }
       case self::LOGIN_FAILED:
         echo "login failed";
         break;
       case self::LOGIN_CANCELED:
         $connection = null;
         echo "LOGIN_CANCELED";
         break;
```

```
            default:
              echo "Bug: unexpected login status: ".$loginStatus;
          }
        }

        $this->logg("$METHOD 2/2: path = $newpath, connection = ");
        $this->logg($connection);
        $this->logg('<br/>');
        return $connect;
    }
}

/*=======================================*\
                                    POST
\*=======================================*/

  public function post()
    //throws JapsClientException, IOException, MalformedURLException
  {
     $numargs = func_num_args();
     if($numargs == 2)  //post($path, $params)
     {
        $path = func_get_arg(0);
        $params = func_get_arg(1);
        return $this->post($path,$params, true);
     }
     elseif($numargs == 3)  //post($path, $params, $loginIfNecessary)
     {
        $path = func_get_arg(0);
        $params = func_get_arg(1);
        $loginIfNecessary = func_get_arg(2);

        $newpath = $this->cocoon.$path; // add "cocoon/" to path

        $METHOD = "post(String,Array,Boolean)";
        $this->logg($METHOD." 1/2: path = ".$newpath.", params = "
                    .$this->paramsToString($params)."., loginIfNecessary = ");
        $this->logg($loginIfNecessary ? 'true<br/>' : 'false<br/>');

        // Prepare connection:
        $connection = new HttpURLConnection($this->urlPrefixWithoutPort,
                                            $this->urlPort,false,
                                            "JapsClient/PHP");
        // Prepare params:
        $params = $this->url_Encode($params);
        // Connect and add Cookies:
        $connect = $connection->post($newpath,$params,$this->addCookies());
        //$this->logg(var_dump($connect));
        // Check response code:
        /*if( $connect["head"]["code"] == 200)  // No Error
        {
          return; //sonst wird login-Abschnitt ausgefuehrt!!!
        }*/
        if( $connect["head"]["code"] >= 400)  // Error
        {
          echo "Could not connect";
          return; //sonst wird login-Abschnitt ausgefuehrt!!!
        }
```

```
        // If necessary, handle cookies:
        $this->handleRecievedCookies($connect);

        // Redirect if necessary:
        if(  $connect["head"]["code"] >= 300 && $connect["head"]["code"] < 400)
        // Redirected
        {
           // Follow redirect (by Http Get):
           $location = $connect["head"]["location"]["uri"];
           $params2 = $connect["head"]["location"]["parameters"];
           $connect = $this->get($this->getPath($location), $params2, false);

           // Check response code again; fail if not OK:
           if( $connect["head"]["code"] != 200)  // Code 200: HTTP_OK?
           {  echo "Could not connect";
              return; //otherwise the login would be executed!!!
           }
        }

        // Login if necessary:
        if ( $loginIfNecessary && $this->checkLoginRequired($connect))
        {
           $loginStatus = $this->login();

           switch ( $loginStatus )
           {
              case self::LOGIN_SUCCESSFUL:
              {
                 $connect = $this->post($path, $params, false);
                 if($this->checkLoginRequired($connect))
                   echo "rejected after successful login";
                 break;
              }
              case self::LOGIN_FAILED:
                echo "login failed";
                break;
              case self::LOGIN_CANCELED:
                $connection = "";
                echo "LOGIN_CANCELED";
                break;
              default:
                echo "Bug: unexpected login status: ".$loginStatus;
           }
        }

        $this->logg($METHOD." 2/2: path = $newpath, connection = ");
        $this->logg($connection);
        return $connect;
    }
}
```

```
  /**
   * <p>
   *   Returns a connection for <code>path</code> via the HTTP Post method.
   *   Can be used for
   *   file uploads. See
   *   <a href="http://www.ietf.org/rfc/rfc1867.txt">RFC 1867:
   *   Form-based File Upload in HTML</a>
   *   and <a href="http://www.ietf.org/rfc/rfc2046.txt">RFC 2046:
   *   MIME Part Two: Media Types</a>
   *   for details.
   * </p>
   * <p>
   *   The content of <code>file</code> is written to the entity body in the
   *   multipart/form-data format. If <code>file</code> is <code>null</code>,
   *    this is
   *   suppressed. <code>fileContentType</code> should be the MIME type of
   *   <code>file</code>.
   * </p>
   * <p>
   *   If <code>loginIfNecessary</code> is <code>true</code>,
   *   the user is logged-in if the
   *   server requires so. This is done by {@link #login()}
   *   (which normally allowes several
   *   tries to log-in). If the login succeded, the request is repeated.
   *   If the server
   *   requires login again (which should not happen in normal operation),
   *   an excehttp://www.tutorials.de/forum/php-tutorials/226367-http-ohne-curl-
   *   version-1-1-a.htmlption is
   *   thrown. If the login failed, an exception is thrown.
   *   If the login was canceled by the
   *   user, no exception is thrown and <code>null</code> is returned.
   * </p>
   */

/*=======================================*\
                    POST-FOR-FILE-UPLOAD
\*=======================================*/

  public function postForFileUpload ()
    //throws JapsClientException, IOException, FileNotFoundException,
    //MalformedURLException
  {
     $numargs = func_num_args();
     if($numargs == 2)  //postForFileUpload ($path, $filename)
     {
        $path = func_get_arg(0);
        $filename = func_get_arg(1);
        $fileContentType = $this->guessContentType($filename);
        if($fileContentType == null)
          $fileContentType = self::DEFAULT_CONTENT_TYPE;

        return $this->postForFileUpload($path, $filename, $fileContentType);

     }
     elseif($numargs == 3)  //postForFileUpload ($path, $filename,
                            //$fileContentType)
     {
        $path = func_get_arg(0);
        $filename = func_get_arg(1);
```

```
   $fileContentType = func_get_arg(2);
   return $this->postForFileUpload($path, $filename, $fileContentType,
                                   true);
}
elseif($numargs == 4)  //postForFileUpload ($path, $filename,
                                            $fileContentType,
                                            $loginIfNecessary)
{
   $path = func_get_arg(0);
   $filename = func_get_arg(1);
   $fileContentType = func_get_arg(2);
   $loginIfNecessary = func_get_arg(3);

   $newpath = $this->cocoon.$path; // add "cocoon/" to path

   $METHOD = "post(String,File,String,Boolean)";
   $this->logg($METHOD." 1/2: path = ".$newpath.", file = ".$filename.",
               fileContentType = $fileContentType, loginIfNecessary = ");
   $this->logg($loginIfNecessary ? 'true<br/>' : 'false<br/>');

   // Prepare connection:
   $connection = new HttpURLConnection($this->urlPrefixWithoutPort,
                                       $this->urlPort,false,
                                       "JapsClient/PHP");
   // Connect and add Cookies:
   $connect = $connection->post($newpath, false, $this->addCookies(),
                                "file=$filename", $fileContentType);

   // Check response code:
   if( $connect["head"]["code"] >= 400)  // Error
   {
     echo "Could not connect";
     return; //otherwise the login would be executed!!!
   }

   // If necessary, handle cookies:
   $this->handleRecievedCookies($connect);

   // Redirect if necessary:
   if(  $connect["head"]["code"] >= 300 && $connect["head"]["code"] < 400)
   // Redirected
   {
      // Follow redirect (by Http Get):
      $location = $connect["head"]["location"]["uri"];
      $connect = $this->get($this->getPath($location), $params = array(),
                            false);

      // Check response code again; fail if not OK:
      if( $connect["head"]["code"] != 200)  // Code 200: HTTP_OK?
      {  echo "Could not connect";
         return; //otherwise the login would be executed!!!
      }
   }

   // Login if necessary:
   if ( $loginIfNecessary && $this->checkLoginRequired($connect))
   {
      $loginStatus = $this->login();
```

```
            switch ( $loginStatus )
            {
               case self::LOGIN_SUCCESSFUL:
               {
                  $connect = $this->postForFileUpload($path, $filename,
                                                      $fileContentType, false);
                  if($this->checkLoginRequired($connect))
                    echo "rejected after successful login";
                  break;
               }
               case self::LOGIN_FAILED:
                 echo "login failed";
                 break;
               case self::LOGIN_CANCELED:
                 $connection = "";
                 echo "LOGIN_CANCELED";
                 break;
               default:
                 echo "Bug: unexpected login status: ".$loginStatus;
            }
         }

         $this->logg($METHOD." 2/2: path = $newpath, connection = ");
         $this->logg($connection);
         return $connect;
    }
}

  //
------------------------------------------------------------------------------
  // Login
  //
------------------------------------------------------------------------------

  /*
   * Performs a login dialog. Returns a {@link LoginDialogResult} object, or
   * <code>null</code> if the user has canceled the dialog.
   *  <code>afterFailure</code>
   * should be set to <code>true</code> if the dialog immediately
   * follows a failed try to
   * login.
   */

  public abstract function performLoginDialog ($afterFailure);
    //throws JapsClientException;

  /*
   * Tries to login with <code>account</code> and <code>password</code>.
   * If succeeded,
   * returns {@link #LOGIN_SUCCESSFUL}, otherwise {@link #LOGIN_FAILED}.
   */
/*
  /*
   * Tries to login with the account and password given by <code>result</code>.
   * If succeeded,
   * returns {@link #LOGIN_SUCCESSFUL}, otherwise {@link #LOGIN_FAILED}.
   */
```

```
/*
 * Tries to login. If the login fails, tries again and again, but at most
 * {@link #maxLoginTries} times. Returns {@link #LOGIN_SUCCESSFUL}
 * if the last login try
 * succeeded, {@link #LOGIN_FAILED} if the last login try failed, and
 * {@link #LOGIN_CANCELED} if the user has canceled the login dialog.
 */

public function login ()
  //throws JapsClientException
{
  $numargs = func_num_args();
  if ($numargs == 0)
  {
    $METHOD = "login()";
    $this->logg($METHOD." 1/2 <br/>");

    $count = 1;
    $status = -1;

    while( $status != self::LOGIN_SUCCESSFUL && $status
            != self::LOGIN_CANCELED && ( $count <= $this->maxLoginTries
            || $this->maxLoginTries == self::UNLIMITED ) )
    {
      $result = $this->performLoginDialog( $status == self::LOGIN_FAILED );
      $status = (is_null($result) ? self::LOGIN_CANCELED
                      : $this->login($result));
      $count++;
    }
  }
  elseif ($numargs == 1)
  { //login(LoginDialogResult result)

    $result = func_get_arg(0);
    return $this->login($result->getAccount(), $result->getPassword());
  }

  elseif ($numargs == 2)
  {
    //login(account,password)
    $account = func_get_arg(0);
    $password = func_get_arg(1);
    $params = array();

    $METHOD = "login(String,char[])";
    $this->logg("$METHOD 1/2: account = $account, password = "
                .$this->hidePassword($password)."<br/>");

  //try
    //{
      // Set request parameters:
      $params[self::ACCOUNT_PARAM] = $account;
      $params[self::PASSWORD_PARAM] = $password;
      $params[self::RESOURCE_PARAM] = $this->composeURL(self::RESOURCE_VALUE);

      // Do post request:
      $connect = $this->post(self::LOGIN_PATH, $params, false);
```

```
        // Login status:
        $status =  ($this->checkLoginRequired($connect) ? self::LOGIN_FAILED
                        : self::LOGIN_SUCCESSFUL);
    }
    else
    {  echo "Too many arguments";  }

    $this->logg("$METHOD 2/2: status = $status <br/>");
    return $status;

  }

  //
-------------------------------------------------------------------------------
  // Maximal number of login tries
  //
-------------------------------------------------------------------------------

  /*
   * Returns the maximum number of login tries. A value of {@link #UNLIMITED}
   * (<code>-1</code>) means the number is not limited. Default:
   *  {@link #UNLIMITED}.
   */

  function getMaxLoginTries ()
  {
     return $this->maxLoginTries;
  }

  /*
   * Sets the maximum number of login tries. A value of
   * {@link #UNLIMITED} (<code>-1</code>)
   * means the number of tries is not limited. Default: {@link #UNLIMITED}.
   * @param maxLoginTries The new maximum number of login tries.
   * @throws IllegalArgumentException If <code>maxLoginTries</code> is not
   *  positive and not
   * equal to {@link #UNLIMITED}.
   */

  function setMaxLoginTries ($maxLoginTries)
  {
     if ( $maxLoginTries <= 0 && maxLoginTries != "UNLIMITED" )
     {  throw new Exception("Invalid value for maximum number of login tries:"
                             .$this->maxLoginTries);  }
     $this->maxLoginTries = $maxLoginTries;
  }
```

```
  //
-------------------------------------------------------------------------------
  // URLs
  //
-------------------------------------------------------------------------------

  /*
   * Returns the prefix of the Japs server.
   */

  function getUrlPrefix()
  {
     return $this->urlPrefix;
  }

  /*
   * Returns the (absolute) URL for a given (relative) path.
   */

  function composeURL($path)
  {
     if ( substr($path,0,0) != "/" )
     {  $path = "/".$path;  }
     return $this->urlPrefix.$path;
  }

  /*
   * Returns the (relative) path for a given (absolute) URL.
   */

  function getPath($url)
  {
     if ( $url == $this->urlPrefix)
     {  return "";  }
     $start = $this->urlPrefix."/";
     if ( strpos($url,$start) != 0 )
     {  throw new Exception("Improper url: ".url." (wrong prefix)"); }
     $start_len = strlen($start);
     return substr($url,$start_len);
  }
```

```
  //
-------------------------------------------------------------------------------
  // Content types
  //
-------------------------------------------------------------------------------

  /**
   * <p>
   *   Returns the content type of <code>filename</code>.
   *   The content type is determined
   *   from the suffix of <code>filename</code>. The suffix is looked up in
   *   {@link #contentTypes}, and the corresponding value is returned.
   * </p>
   * <p>
   *   If <code>filename</code> has no suffix,
   *   or if no entry for the suffix is found in
   *   {@link #contentTypes}, <code>null</code> is returned.
   * </p>
   */

   function guessContentType($filename)
   {
     if(!$pos = strrpos($filename,"."))
       return null;
     $suffix = substr($filename,$pos+1);
     return $this->getContentTypeForSuffix($suffix);
   }

  /**
   * Returns the content type for <code>suffix</code>.
   * The suffix is looked up in
   * {@link #contentTypes}. If no entry for the suffix is found,
   *  <code>null</code> is
   * returned.
   */

  function getContentTypeForSuffix($suffix)
  {
     return $this->contentTypes[$suffix];
  }

  /**
   * Sets the content type for <code>suffix</code>.
   */

  function setContentTypeForSuffix($suffix, $contentType)
  {
     $this->contentTypes[$suffix] = $contentType;
  }
```

```
  /**
   * Removes the content type for <code>suffix</code>.
   */

  function removeContentTypeForSuffix($suffix,$contentType)
  {
     $this->contentTypes[$suffix] = "";
  }

  /**
   * Returns all suffixes for which content types are specified.
   */

  function getContentTypeSuffixes()
  {
     return $this->contentTypes;
  }

  //
------------------------------------------------------------------------------
  // Utilities to create the request
  //
------------------------------------------------------------------------------

  /**
   * Url-encodes the entries of <code>params</code> and returns
   * the result as a string.
   */
   protected function url_Encode($params)
   {
      if( $params == null)
        return null;
      if(is_string($params))
      {  return $params;  }
      $query_string = array();
      foreach ($params as $key => $value)
      {
        if(is_array($value))
        {
           $string = "";
           foreach($value as $helpvalue)
           {  $string .= $helpvalue;
           }
           $query_string[] = urlencode($key) . '=' . urlencode($string);
        }
        else
        {
           $query_string[] = urlencode($key) . '=' . urlencode($value);
        }
      }
      return implode('&', $query_string);
    }
```

```
  //
-------------------------------------------------------------------------------
  // Logging
  //
-------------------------------------------------------------------------------

  /*
   * <p>$fp
   *   Writes a log message.
   * </p>
   * <p>
   *   This implementation does nothing.
   *   Extending classes can overwrite this to implement a
   *   logging mechanism.
   * </p>
   */

  function logg($message)
  {
     if($this->logFile != "")
     {
        if($fp = fopen($this->logFile, 'a'))
        {
           fputs($fp, self::LOG_PREFIX.': '
                  .date(self::DEFAULT_TIMESTAMP_PATTERN).' => ');
           fputs($fp, $message);
           fputs($fp, "\r\n");
        }
        fclose($fp);
     }
  }


  //
-------------------------------------------------------------------------------
  // Analysing the response
  //
-------------------------------------------------------------------------------
  /*
   * Returns <code>true</code> if login is required for
   * <code>connection</code>, otherwise
   * <code>false</code>.
   */

  protected function checkLoginRequired($connect)
  {
     $raw = $connect['head']['raw'];
     if(!strpos($raw,self::LOGIN_REQUIRED_HEADER.':'))
     {  return false;  }
     else
     {
       $start = strpos($raw,self::LOGIN_REQUIRED_HEADER.':') + 24;
       $value = trim(substr($raw,$start,3));
       if ($value == "")
         return false;
       else if ($value == "yes" || $value == "true")
         return true;
       else if ($value == "no" || $value == "false")
         return false;
```

```
      else
        echo "Unknown JapsResponseHeader.LOGIN_REQUIRED - login_required: "
                .$value;
    }
  }

 //
-------------------------------------------------------------------------------
  // Utilities for log messages
  //
-------------------------------------------------------------------------------

  /**
   * Convenience method to replace a password
   * (or another confidential piece of text) by a
   * sequence of asterisks. Used for log messages.
   */

  protected function hidePassword ($password)
  {
     $chars = "";
     for ($i = 0; $i < count($password); $i++)
     {  $chars .= '*';  }

     return $chars;
  }

  /**
   * Convenience method to convert a numeric status code into a self-explanatory
   * string. Used for log messages.
   */

  protected function loginStatusToString ($loginStatus)
  {
    switch ( $loginStatus )
      {
         case self::LOGIN_SUCCESSFUL:
           return "LOGIN_SUCCESSFUL";
         case self::LOGIN_FAILED:
           return "LOGIN_FAILED";
         case self::LOGIN_CANCELED:
           return "LOGIN_CANCELED";
         default:
           return "[unknown status code: " + $loginStatus + "]";
      }
  }

  /**
   * <p>
   *   Convenience method to convert parameters to an informative string.
   *   Used for log
   *   messages.
   * </p>
   * <p>
   *   Map values that are of type <code>char[]</code>
   *   are regarded as confidential and
   *   protected by the {@link #hidePassword} method.
   * </p>
   */
```

```
  protected function paramsToString ($params)
  {
     if ( $params == null )
       return null;
     if (is_string($params))
       return $params;
     $buffer = "{";
     foreach ($params as $key => $value)
     {
        if(is_array($value))
        {
           $string = $this->hidePassword($value);
           $buffer .= $key.'='.$string;
        }
        else
        {
           $buffer .= $key.'='.$value;
        }
        $buffer .= ",";
     }
     $buffer = trim($buffer,',');
     $buffer .= "}";
     return $buffer;
  }


}
?>
```

A1.14 Quellcode JapsSynchronise.class.php

```
<?php

// Including the LoginDialogResult class for performing the login
require_once ('LoginDialogResult.class.php');

class JapsSynchronise extends JapsClient{

/*
* The account for login.
*/

protected $account = null;

/*
* The password for login.
*/

private $password = array();


/**
 * Constructor for JapsSynchronise
 *
 * $serverPath - path to mumies cocoon directory
 * $url - path to the synchronisation method
 * $logFilename - Name of the logfile, not necessary
 * $dbValue - the array with the values to synchronise
 *
 */
public function JapsSynchronise (){
    global $CFG;
    // calling the super classes constructor
    $logFilename = $CFG->syncLogfile;
    parent::__construct($CFG->syncServer, $logFilename);
     $this->setAccount($CFG->syncUser);
     for ($i = 0; $i < strlen($CFG->syncPassword); $i++){
      $password[] = substr ($CFG->syncPassword, $i, 1);
     }
    $this->setPassword($password);
 }

  public function performLoginDialog ($afterFailure)
  {
     if ( $this->account == null || $this->password == null )
       throw new Exception("Account and/or password null");
     return new LoginDialogResult($this->account, $this->password);
  }
```

```
    /**
    * Sets the account.
    */

  public function setAccount ($account)
  {
    if ( $account == null )
      //throw new IllegalStateException("Account null");
     echo "Account null";
    $this->account = $account;
  }

  /**
   * Sets the password.
   */

  public function setPassword ($password)
  {
    if ( $password == null )
      //throw new IllegalStateException("Password null");
     echo "Password null";
    $this->password = $password;
  }
}

?>
```

A1.15 Quellcode LoginDialogResult.class.php

```
<?php
/**
 * <p>
 *   Represents a pair consisting of an account name and password.
 * </p>
*/

class LoginDialogResult
{
  /**
   * The account.
   */

  protected $account = null;

  /**
   * The password.
   */

  private $password = array();

  /**
   * Returns the the account. (Cf. {@link #account account}.)
   */

  public function getAccount ()
  {
    return $this->account;
  }

  /**
   * Returns the password.
   */

  public function getPassword ()
  {
    $password = array();
    foreach($this->password as $key => $value)
    {
      $password[$key] = $value;
    }

    return $password;
  }

  /**
   * Returns the password as a string.
   */

  public function getPasswordAsString ()
  {
    $password = "";
    foreach($this->password as $value)
    {
      $password .= $value;
    }
```

```
    return $password;
  }

  /**
   * Replaces all characters in <code>chars</code> by the null character.
   */

  public static function clear_array ($chars)
  {
    for ($i = 0; $i < count($chars); $i++)
    {  $chars[$i] = 0;  }
  }

  /**
   * Erases the data stored in this object.
   */

  public function clear ()
  {
    $this->account = null;
    if ( !empty($this->password))
      {
        $this->clear_array(&$this->password);
        $this->password = array();
      }
  }

  /**
   * Creates a new <code>LoginDialogResult</code> object.
   */

  public function LoginDialogResult ($account = "", $password = array())
  {
    $this->account = $account;
    $this->password = array();
    foreach($password as $key => $value)
    {
      $this->password[$key] = $value;
    }
  }

}
?>
```

A1.16 Quellcode HttpURLConnection.class.php

```
<?php
/*
httpconnection.class.php
Version 1.1

Part of the PHP class collection
http://www.sourceforge.net/projects/php-classes/
Quelle: http://www.tutorials.de/forum/php-tutorials/226367-http-ohne-curl-
version-1-1-a.html

Written by: Dennis Wronka
License: LGPL
*/
class HttpURLConnection
{
    private $host;
    private $port;
    private $ssl;
    private $useragent;

    //public function __construct($host,$port=80,$ssl=false,$useragent='Java')
    //for MUMIE 1
    public function __construct($host,$port=80,$ssl=false,
                                $useragent='JapsClient/PHP') //fuer MUMIE 2
    {
        $this->host=$host;
        $this->port=$port;
        $this->ssl=$ssl;
        $this->useragent=$useragent;
    }

       public function __toString() //debug()
  {
     $string .= "Object::HttpURLConnection<br>";
     return $string;
  }

    private function decodereply($reply)
    {
        $headend=strpos($reply,"\r\n\r\n")+2;
        $head=substr($reply,0,$headend);
        $httpversion=substr($head,5,3);
        $contentlength='';
        $contentlengthstart=strpos($head,'Content-Length:');
        if ($contentlengthstart!=false)
            {
                $contentlengthstart+=16;
                $contentlengthend=strpos($head,"\r\n",$contentlengthstart);
                $contentlength=substr($head,$contentlengthstart,
                                      $contentlengthend-$contentlengthstart);
            }
        if ($httpversion=='1.0')
            {
                $datastart=$headend+2;
                $body=substr($reply,$datastart,strlen($reply)-$datastart);
            }
```

```
elseif ($httpversion=='1.1')
    {
        $encoding='';
        $encodingstart=strpos($head,'Transfer-Encoding:');
        if ($encodingstart!=false)
            {
                $encodingstart+=19;
                $encodingend=strpos($head,"\r\n",$encodingstart);
                $encoding=substr($head,$encodingstart,
                                 $encodingend-$encodingstart);
            }
        if ($encoding=='chunked')
            {
                $datasizestart=$headend+2;
                $datasizeend=strpos($reply,"\r\n",$datasizestart);
                $datasize=hexdec(trim(substr($reply,$datasizestart,
                                        $datasizeend-$datasizestart)));
                $body='';
                while ($datasize>0)
                    {
                        $chunkstart=$datasizeend+2;
                        $body.=substr($reply,$chunkstart,$datasize);
                        $datasizestart=$chunkstart+$datasize+2;
                        $datasizeend=strpos($reply,"\r\n",
                                            $datasizestart);
                        $datasize=hexdec(trim(substr($reply,
                                            $datasizestart,
                                        $datasizeend-$datasizestart)));
                    }
            }
        else
            {
                $datastart=$headend+2;
                $datasize=$contentlength;
                $body=substr($reply,$datastart,$datasize);
            }
    }
$code=substr($head,9,3);
$serverstart=strpos($head,'Server:')+8;
$serverend=strpos($head,"\r\n",$serverstart);
$server=substr($head,$serverstart,$serverend-$serverstart);
$contenttype='';
$contenttypestart=strpos($head,'Content-Type:');
if ($contenttypestart!=false)
    {
        $contenttypestart+=14;
        $contenttypeend=strpos($head,"\r\n",$contenttypestart);
        $contenttype=substr($head,$contenttypestart,
                            $contenttypeend-$contenttypestart);
    }
$location='';
$locationstart=strpos($head,'Location:');
if ($locationstart!=false)
    {
        $locationstart+=10;
        $locationend=strpos($head,"\r\n",$locationstart);
        $location=substr($head,$locationstart,
                         $locationend-$locationstart);
        $location_array=explode('?',$location);
```

```
        $parameters='';
        if (isset($location_array[1]))
            {
                $parameters=$location_array[1];
            }
        $location=array('uri'=>$location_array[0],
                        'parameters'=>$parameters);
        if (empty($parameters))
            {
                unset($location['parameters']);
            }
    }
$cookies=array();
$cookiestart=strpos($head,'Set-Cookie:');
while ($cookiestart!=false)
    {
        $cookiestart+=12;
        $cookieend=strpos($head,"\r\n",$cookiestart);
        $cookie=substr($head,$cookiestart,$cookieend-$cookiestart);
        $cookie_array=explode(';',$cookie);
        $expirydate='';
        $path='';
        for ($x=0;$x<count($cookie_array);$x++)
            {
                $cookie_array[$x]=explode("=",$cookie_array[$x]);
                if ($x==0)
                    {
                        $name=$cookie_array[$x][0];
                        $value=$cookie_array[$x][1];
                    }
                else
                    {
                        if (trim($cookie_array[$x][0])=='expires')
                            {
                                $expirydate=array('string'
                                                  =>$cookie_array[$x][1],
                                                   'timestamp'=>strtotime
                                                  ($cookie_array[$x][1]));
                            }
                        elseif (trim($cookie_array[$x][0])=='Path')
                            {
                                $path=$cookie_array[$x][1];
                            }
                    }
            }
        $cookie=array('name'=>$name,'value'=>$value,'Path'=>$path,
                      'expirydate'=>$expirydate);
        if (empty($path))
            {
                unset($cookie['Path']);
            }
        if (empty($expirydate))
            {
                unset($cookie['expirydate']);
            }
        $cookies[]=$cookie;
        $cookiestart=strpos($head,'Set-Cookie:',$cookieend);
    }
```

```
        $headdata=array('raw'=>$head,'httpversion'=>$httpversion,
                        'code'=>$code,'server'=>$server,
                        'contentlength'=>$contentlength,
                        'contenttype'=>$contenttype,'location'=>$location,
                        'cookies'=>$cookies);
        if ((empty($contentlength)) && ($contentlength!=0))
            {
                unset($headdata['contentlength']);
            }
        if (empty($contenttype))
            {
                unset($headdata['contenttype']);
            }
        if (empty($location))
            {
                unset($headdata['location']);
            }
        if (empty($cookies))
            {
                unset($headdata['cookies']);
            }
        $data=array('head'=>$headdata,'body'=>$body);
        return $data;
    }

/*=================================================*\
#                                                   HEAD
#
\*=================================================*/

    public function
head($uri='/',$parameters=false,$cookies=false,$authuser='',$authpassword='')
    {
        if ($this->ssl==true)
            {
                $connection=@fsockopen('ssl://'.$this->host,$this->port);
            }
        else
            {
                $connection=@fsockopen($this->host,$this->port);
            }
        if ($connection==false)
            {
                return false;
            }
        if ((empty($uri)) || ($uri{0}!='/'))
            {
                $uri='/'.$uri;
            }
        if (($parameters!=false) && (!empty($parameters)))
            {
                $paramstring='?'.$parameters;
            }
        else
            {
                $paramstring='';
            }
        if (($cookies!=false) && (!empty($cookies)))
            {
```

```
                $cookiestring='Cookie: '.$cookies."\r\n";
            }
        else
            {
                $cookiestring='';
            }
        if (!empty($authuser))
            {
                $authstring='Authorization: Basic '
                            .base64_encode($authuser.':'.$authpassword)."\r\n";
            }
        else
            {
                $authstring='';
            }
        $host=$this->host;
        if ($this->port!=80)
            {
                $host.=':'.$this->port;
            }
        fwrite($connection,'HEAD '.$uri.$paramstring
                ." HTTP/1.1\r\nHost: ".$host.
                "\r\nUser-Agent: "
                .$this->useragent."\r\n".$cookiestring
                .$authstring."Connection: close\r\n\r\n");
        $reply='';
        while (!feof($connection))
            {
                $reply.=@fread($connection,128);
            }
        fclose($connection);
        $data=$this->decodereply($reply);
        return $data;
    }

/*=====================================================*\
#                                                 GET
#
\*=====================================================*/


    public function get($uri='/',$parameters=false,
                        $cookies=false,$authuser='',$authpassword='')
    {
        if ($this->ssl==true)
            {
                $connection=@fsockopen('ssl://'.$this->host,$this->port);
            }
        else
            {
                $connection=@fsockopen($this->host,$this->port);
            }
        if ($connection==false)
            {       echo "connection==false";
                return false;
            }
        if ((empty($uri)) || ($uri{0}!='/'))
            {
                $uri='/'.$uri;
```

```
            }
        if (($parameters!=false) && (!empty($parameters)))
            {
                $paramstring='?'.$parameters;
            }
        else
            {
                $paramstring='';
            }
        if (($cookies!=false) && (!empty($cookies)))
            {
                $cookiestring='Cookie: '.$cookies."\r\n";
            }
        else
            {
                $cookiestring='';
            }
        if (!empty($authuser))
            {
                $authstring='Authorization: Basic '
                            .base64_encode($authuser.':'.$authpassword)."\r\n";
            }
        else
            {
                $authstring='';
            }
        $host=$this->host;
        if ($this->port!=80)
            {
                $host.=':'.$this->port;
            }
        fwrite($connection,'GET '.$uri.$paramstring." HTTP/1.1\r\nHost: "
                  .$host."\r\nUser-Agent: "
                  .$this->useragent."\r\n"
                  .$cookiestring.$authstring."Connection: close\r\n\r\n");
        $reply='';
        while (!feof($connection))
            {
                $reply.=@fread($connection,128);
            }
        fclose($connection);
        $data=$this->decodereply($reply);
        return $data;
    }
/*==================================================*\
#                                          POST
#
\*==================================================*/
    public function post($uri='/',$parameters=false,
                         $cookies=false,$fileparameters=false,
                         $mimetypes=false,$authuser='',$authpassword='')
    {
        if ($this->ssl==true)
            {
                $connection=@fsockopen('ssl://'.$this->host,$this->port);
            }
        else
            {
                $connection=@fsockopen($this->host,$this->port);
```

```
    }
if ($connection==false)
    {
        return false;
    }
if ((empty($uri)) || ($uri{0}!='/'))
    {
        $uri='/'.$uri;
    }
if (($cookies!=false) && (!empty($cookies)))
    {
        $cookiestring='Cookie: '.$cookies.'\r\n';
    }
else
    {
        $cookiestring='';
    }
if (!empty($authuser))
    {
        $authstring='Authorization: Basic '
                    .base64_encode($authuser.':'.$authpassword)."\r\n";
    }
else
    {
        $authstring='';
    }
$host=$this->host;
if ($this->port!=80)
    {
        $host.=':'.$this->port;
    }
if (($fileparameters==false) || (empty($fileparameters)))
    //keine Datei spezifiziert
    {
        if (($parameters!=false) && (!empty($parameters)))
          //nur Parameter senden
            {
                $contentlength=strlen($parameters);
                fwrite($connection,'POST '.$uri
                        ." HTTP/1.1\r\nHost: ".$host
                        ."\r\nUser-Agent: "
                        .$this->useragent."\r\n".
                        $cookiestring.$authstring
                        ."Connection: close\r\n");
                fwrite($connection,
                      "Content-Type:
                   application/x-www-form-urlencoded\r\nContent-Length:"
                   .$contentlength."\r\n\r\n".$parameters);
        }
        else
            {
                fwrite($connection,'POST '.$uri
                        ." HTTP/1.1\r\nHost: ".$host.
                         "\r\nUser-Agent: ".$this->useragent
                        ."\r\n".$cookiestring
                        .$authstring."Connection: close\r\n\r\n");
            }
    }
```

```
else  //a file was spezified
    {
        $params=explode('&',$parameters);
        for ($x=0;$x<count($params);$x++)
            {
                $params[$x]=explode('=',$params[$x]);
            }
        $fileparams=explode('&',$fileparameters);
        for ($x=0;$x<count($fileparams);$x++)
            {
                $fileparams[$x]=explode('=',$fileparams[$x]);
            }
        if (($mimetypes!=false) && (!empty($mimetypes)))
            {
                $mimetypeparams=explode(',',$mimetypes);
            }
        if (!isset($mimetypeparams))
            {
                $mimetypeparams=array();
            }
        while (count($mimetypeparams)<count($fileparams))
            {
                $mimetypeparams[]='application/octet-stream';
            }
        $boundary='-----------------------'.substr(md5(uniqid()),0,15);
        $content='';
        for ($x=0;$x<count($fileparams);$x++)
            {
                $postfile=fopen($fileparams[$x][1],'r');
                $filecontent=fread($postfile,
                                   filesize($fileparams[$x][1]));
                fclose($postfile);
                $content.='--'.$boundary."\r\n";
                $content.='Content-Disposition: form-data;
                               name="'.$fileparams[$x][0].'";
                               filename="'.$fileparams[$x][1]
                               .'"'."\r\n";
                $content.='Content-Type: '
                          .$mimetypeparams[$x]."\r\n\r\n";
                $content.=$filecontent."\r\n";
            }
        for ($x=0;$x<count($params);$x++)
            {
                $content.='--'.$boundary."\r\n";
                $content.='Content-Disposition: form-data;
                           name="'.$params[$x][0].'"'."\r\n\r\n";
                if (!empty($params[$x][1]))
                    {
                        $content.=$params[$x][1]."\r\n";
                    }
            }
        $content.='--'.$boundary."--\r\n";
        $contentlength=strlen($content);
        fwrite($connection,'POST '.$uri." HTTP/1.1\r\nHost: ".$host.
                "\r\nUser-Agent: "
                .$this->useragent."\r\n".$cookiestring
                .$authstring."Connection: close\r\n");
```

```
                fwrite($connection,'Content-Type: multipart/form-data;
                        boundary='.$boundary
                          ."\r\nContent-Length: ".$contentlength."\r\n\r\n");
                fwrite($connection,$content,$contentlength);
            }
        $reply='';
        /*while (!feof($connection))
            {
                $temp_reply = @fread($connection,1024);
                $ascii = ord(strrev($temp_reply));
                //$temp_reply = fgets($connection,128);
                $reply.= $temp_reply;
            }*/
        $reply .= @fread($connection, 1024);
        fclose($connection);
        $data=$this->decodereply($reply);
        return $data;
    }
}
?>
```

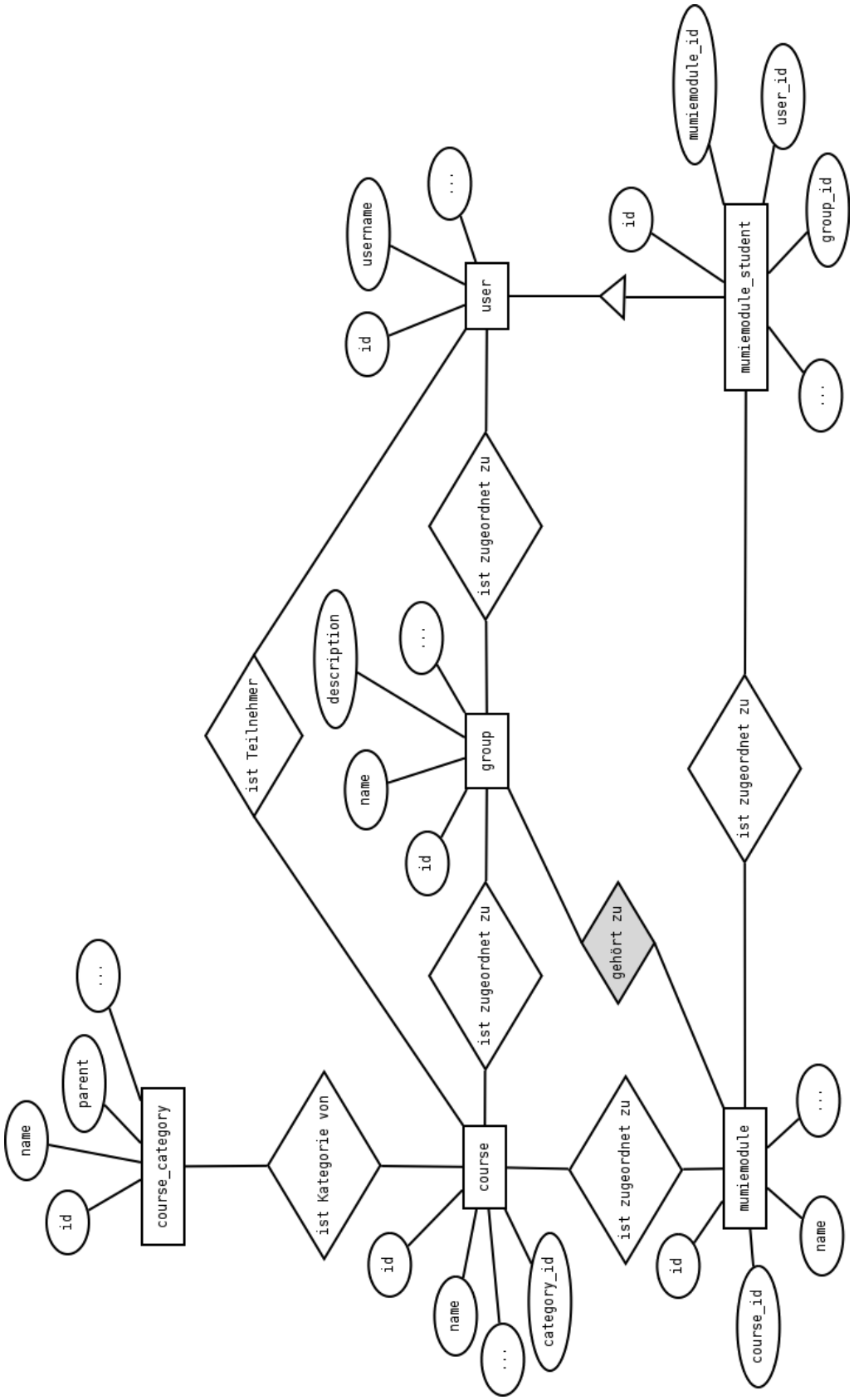
A1.17 Englische Sprachdatei der MUMIE Aktivität

```
<?PHP // $Id: mumiemodule.php,v 1.4 2006/11/01 09:44:05 moodler Exp $
      // mumiemodule.php - created by PR for Moodle 1.9 alpha


$string['alreadyused'] = 'The MUMIE-course you want to link with this activity
                          is already used for another one. \n' .
                         'If you go on, you will change that mapping
                          which might cause problems. \n'.
                         'Do you want to continue proceeding?';
$string['error'] = 'Sorry, an unknown error has occured.';
$string['description'] = 'Description';
$string['entermumie'] = 'Enter MUMIE';
$string['grade'] = 'Points';
$string['justonegroup'] = 'One or more students are in more than one group in
                        this course. This is forbidden when using MUMIE-module';
$string['justonegroupsingle'] = 'A student must not be in more than one group!
                                 This is forbidden when using MUMIE-module';
$string['justonemod'] = 'It is not allowed to have more than one
                         MUMIE-activities within one course!';
$string['mumie'] = '-- MUMIE - A mathematical Learning Environment --';
$string['mumie:manage'] = 'Manage MUMIE activities';
$string['mumie:participate'] = 'Participate in MUMIE activities';
$string['lastmodified'] = 'Last worked on';
$string['modulename'] = 'MUMIE-activity';
$string['modulenameplural'] = 'MUMIE-activities';
$string['notingroupstudent'] = 'Sorry, but you need to be part of a group
                                 to see this module.';
$string['notingroup'] = 'Sorry, but you need to be teacher of a group
                         to see this grades.';
$string['nosettings'] = 'All the MUMIE settings have not been set up!
                         Please contact your administrator.';
$string['showgrades'] = 'Show students grades';
$string['syncPassword'] = 'MUMIE\'s Sync-Password';
$string['syncServer'] = 'The basic URI to the MUMIE';
$string['syncUser'] = 'login-name of MUMIE\'s sync-user';
$string['toomuchtutors'] = 'There are groups with more than one members
                            that are regarded as tutors.
                           This is not allowed for MUMIE. Please change first.';
$string['toomuchtutorssingle'] = 'A group must not include more than one users
                                  with tutor-capabilities! This is forbidden
                                  when using MUMIE-module';
$string['visibletostudents'] = 'Show activity to students';
$string['nogroup'] = 'There are no groups defined for the course.';
$string['mumiecourse'] = 'Mumie-Course';
$string['warning'] = 'Warning';

?>
```

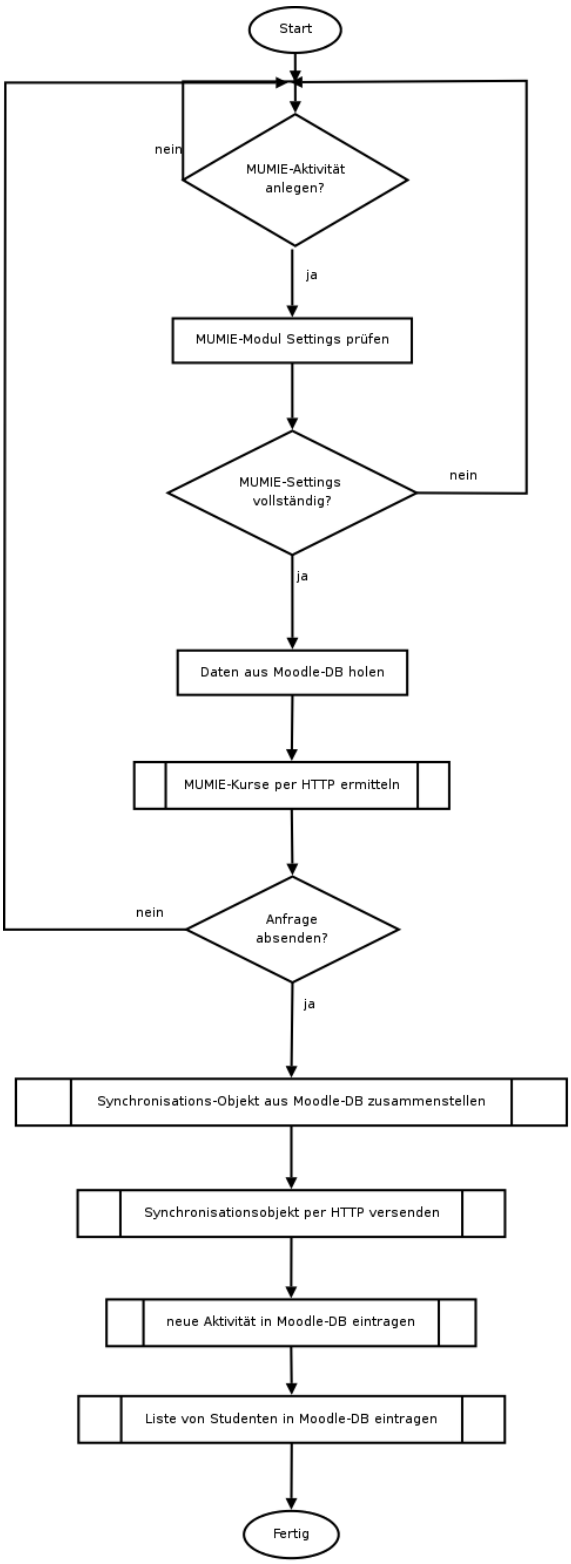
A1.18 Deutsche Sprachdatei der MUMIE Aktivität

```
<?PHP // $Id: mumiemodule.php,v 1.4 2006/11/01 09:44:05 moodler Exp $
      // mumiemodule.php - created by PR for Moodle 1.9 alpha


$string['alreadyused'] = 'Der MUMIE-Kurs, den Sie mit dieser Aktivität verbinden
                          möchten, wird schon in einer anderen genutzt. \n' .
                         'Wenn Sie fortfahren, ändern Sie diese Zuordnung, was
                          Probleme verursachen kann. \n'.
                         'Wollen Sie dennoch fortfahren?';
$string['error'] = 'Ein unbekannter Fehler ist aufgetreten.';
$string['description'] = 'Beschreibung';
$string['entermumie'] = 'MUMIE betreten';
$string['grade'] = 'Punkte';
$string['justonegroup'] = 'Mindestens ein Student in dieser Lehrveranstaltung
                           ist Teilnehmer in mehreren Gruppen. \n'.
                           'Das ist bei der Verwendung einer MUMIE Aktivität
                           nicht erlaubt';
$string['justonegroupsingle'] = 'Ein Student darf nicht in mehr als einer Gruppe
                                 Teilnehmen. \n'.
                                'Das ist bei der Verwendung einer MUMIE
                                 Aktivität nicht erlaubt.';
$string['justonemod'] = 'Es darf nur maximal eine MUMIE Aktivität pro
                         Lehrveranstaltung existieren!';
$string['mumie'] = '-- MUMIE - Eine mathematische Lernumgebung';
$string['mumie:manage'] = 'MUMIE Aktivitäten bearbeiten';
$string['mumie:participate'] = 'In MUMIE Aktivitäten teilnehmen';
$string['lastmodified'] = 'Zuletzt bearbeitet';
$string['modulename'] = 'MUMIE Aktivität';
$string['modulenameplural'] = 'MUMIE Aktivitäten';
$string['notingroupstudent'] = 'Sorry, Sie müssen Teilnehmer einer Gruppe sein,
                                um diese Aktivität zu sehen.';
$string['notingroup'] = 'Sorry, Sie müssen Trainer einer Gruppe sein, um die
                         Noten zu sehen.';
$string['nosettings'] = 'Die MUMIE Grundeinstellungen sind unvollständig. Bitte
                         kontaktieren Sie einen Administrator.';
$string['showgrades'] = 'Zeige Punktzahlen der Studenten';
$string['syncPassword'] = 'MUMIE\'s Synchronisationspasswort';
$string['syncServer'] = 'Die Basis-URL der MUMIE';
$string['syncUser'] = 'Nutzernamen des Synchronisationsnutzers in MUMIE';
$string['toomuchtutors'] = 'Es gibt Gruppen mit mehr als einem Teilnehmer mit
                            Trainerrechten. \n'.
                           'Dies ist bei Verwendung einer MUMIE Aktivität nicht
                            erlaubt. Bitte ändern Sie dies zuerst.';
$string['toomuchtutorssingle'] = 'Eine Gruppe darf nicht mehr als einen Nutzer
                                  mit Trainerrechten beinhalten! \n'.
                                  'Das ist bei der Verwendung einer MUMIE
                                  Aktivität nicht erlaubt.';
$string['visibletostudents'] = 'Aktivität für Studenten anzeigen';
$string['nogroup'] = 'Es existieren keine Gruppen in dieser Lehrveranstaltung.';
$string['mumiecourse'] = 'MUMIE-Kurs';
$string['warning'] = 'Warnung';

?>
```

A2 ER-Diagramm der wichtigen Tabellen der Moodle DB



A3 Flussdiagramm der internen Verarbeitung in Moodle



A4.1 Use-Case: Erstellen eines Moodle-Kurses mit MUMIE Aktivität

Name	Erstellen eines Moodle-Kurses mit MUMIE Aktivität
Zusammenfassung	Ein Dozent (alt. Admin) geht auf Moodle, um dort einen Kurs mit eingebetteter LEMUREN-Aktivität zu erstellen.
Akteure	Dozent / Assistent (alternativ Admin), MUMIE Aktivität, MUMIE
Auslöser	Eine Moodle Lehrveranstaltung mit Verbindung zur MUMIE soll erstellt werden
Annahme	Der Nutzer steht mit entsprechenden Rechten in der Moodle Datenbank
Vorbedingungen	- Innerhalb der MUMIE existieren bereits MUMIE-Kurse. - Die MUMIE Aktivität ist in Moodle integriert und vollständig konfiguriert (URI zur MUMIE, Login-daten des LMS in MUMIE) - Der Moodle-Kurs wurde bereits erstellt (inkl. Gruppen und Studenten)
Verlaufsschritte	1. Der Dozent bewegt sich (authentifiziert) auf Moodle und den entsprechenden Kurs. 2. Auf der Kurs-Startseite wählt der Dozent „Turn editing on“ 3. Unter „insert activity“ wählt er aus dem Pull-Down-Menu „MUMIE activity“ 4. Das Modul schickt eine Anfrage nach allen Lerninhalten an MUMIE. 5. MUMIE antwortet mit einer Liste aller MUMIE-Kurse inklusive ID und bereits verbundenen Lehrveranstaltungen. 6. Auf der Einstellungsseite der Aktivität trägt der Dozent die entsprechenden Informationen ein (Titel, Beschreibung) und wählt aus dem Pull-Down-Menu mit den MUMIE-Kursen den für diese Lehrveranstaltung entsprechenden aus und wählt „save“ 7. Moodle sucht alle relevanten Informationen zusammen (Course-category, course, groups, teacher, students), strukturiert diese den Synchronisationsbefehlen entsprechend und schickt diese per HTTP an MUMIE 8. Das Modul schreibt die relevanten Informationen (course, users, MUMIE-Kurs) in die Moodle Datenbank.
Nachbedingung	- Innerhalb des Moodle-Kurses existiert nun die MUMIE Aktivität mit Verbindung zur MUMIE - in die MUMIE wurden die relevanten Informationen in die Datenbank eingetragen

A4.2 Use-Case: Nachträgliches Hinzufügen von Studenten

Name	Nachträgliches Hinzufügen von Studenten zu einer Moodle Lehrveranstaltung mit MUMIE Aktivität
Zusammenfassung	Ein Student wird nachträglich zu einer Moodle Lehrveranstaltung hinzugefügt, in welchem bereits eine MUMIE Aktivität existiert.
Akteure	Dozent (oder Student selbst),Moodle, MUMIE Aktivität, MUMIE
Auslöser	Ein (oder mehrere) Studenten müssen noch in eine Lehrveranstaltung eingefügt werden
Annahme	Der Nutzer steht mit entsprechenden Rechten in der Moodle Datenbank (für den Fall, dass der Student sich selbst einschreibt, muss dies für den Kurs erblaubt sein)
Vorbedingung	- Der Moodle-Kurs mit eingebetteter MUMIE Aktivität existiert in Moodle
Verlaufsschritte	1. Der Dozent wählt in den Einstellungen des Moodle-Kurses über „assign roles“ einen oder mehrere Studenten aus und fügt diese zu oder Der Student wählt von der Moodle-Startseite den entsprechenden Moodle-Kurs und wählt bei der Frage, ob er sich für den Kurs einschreiben möchte, „ja“ 2. Moodle nimmt die entsprechenden User-Daten und schreibt die Verbindung der jeweils eingefügten User zum Kurs in die interne DB. 3. Moodle geht alle Module des bearbeiteten Kurses durch und prüft, welche davon über die entsprechende Aktion (hier: User einfügen) informiert werden wollen und ruft die dortige Funktion auf (hier in der MUMIE Aktivität) 4. Die MUMIE Aktivität prüft, in welchem Context der User eingefügt wurde und schreibt eine entsprechend neue Zeile zum Modul in die Moodle Datenbank 5. Die Aktivität sammelt die für die MUMIE relevanten Informationen zusammen (User-Daten, course-Daten) und schickt diese über HTTP als Synchronisationsbefehl an MUMIE.
Nachbedingung	- Der (oder die) Benutzer sind in Moodle sowohl zum entsprechenden Kurs wie auch zur Aktivität hinzugefügt worden - in die MUMIE wurden die relevanten Informationen in die Datenbank eingetragen

Jegliche anderen Änderungen oder Ergänzungen zu Kursen, Kategorien, Usern oder Gruppen funktionieren äquivalent.